ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Ametek / VTI Instruments / VXI Technology EX1000A* at our website: ***Click HERE***

EX1000A/EX1000A-TC

48-CHANNEL PRECISION VOLTAGE MEASUREMENT INSTRUMENT

EX1016A

32-CHANNEL PRECISION VOLTAGE MEASUREMENT,
16-CHANNEL PRECISION THERMOCOUPLE INSTRUMENT

EX1032A

16-CHANNEL PRECISION VOLTAGE MEASUREMENT,
32-CHANNEL PRECISION THERMOCOUPLE INSTRUMENT

EX1048A

48-CHANNEL PRECISION THERMOCOUPLE INSTRUMENT

EX10SC

16-CHANNEL SIGNAL CONDITIONING EXPANSION CHASSIS

EX1000A-51C

51-CHANNEL PRECISION VOLTAGE MEASUREMENT

RX1032

32-CHANNEL PRECISION RUGGED THERMOCOUPLE INSTRUMENT

EX1044A

2-CHANNEL PRECISION VOLTAGE ISOLATED MEASUREMENT
2-CHANNEL PRECISION CURRENT ISOLATED MEASUREMENT
44-CHANNEL PRECISION THERMOCOUPLE INSTRUMENT

EX1044

44-CHANNEL PRECISION THERMOCOUPLE INSTRUMENT, W/4-CH 5B MOD SLOT

USER'S MANUAL

P/N: 82-0122-000
May 04, 2020

AMETEK Programmable Power
9250 Brown Deer Road
San Diego, CA 92121, (858) 450-0085

TABLE OF CONTENTS

INTRODUCTION
About AMETEK 10
Trademarks 10
Exclusion for Documentation 10
Contact Information 10
Certification 11
Warranty Terms 11
Product Return Procedure 11
DECLARATION OF CONFORMITY 12
DECLARATION OF CONFORMITY 13
GENERAL SAFETY INSTRUCTIONS 14
Terms and Symbols 14
Warnings 14
SECTION 1 16
INTRODUCTION 16
Overview 16
Features 17
Combined Thermocouple and Analog Inputs 17
End-to-end Self-calibration 17
Scalable for Synchronized High-Speed, High-Channel Count 17
Open Thermocouple Detection (OTD) 17
Cold Junction Compensation (CJC) 18
Simplified Installation, Setup and Control 18
Channel Independence 18
Measurement Range 18
Hardware Filter 19
Input Connectors 19
Sampling Rate 19
Digital I/O and Limits 19
LXI Trigger Bus 20
Triggering 20
EX10xxA Specifications 20
EX10xxA Dimensional Diagram 25
RX1032 Diagram 27
EX1044A/EX1044 Diagram 29
Explanation of Specifications 31
Maximizing Measurement Performance 32
Utilize self-calibration 32
Dynamic Calibration in RX1032 32
Utilize self-test 32
Allow for cold junction thermal stabilization 32
Select the proper hardware filter 34
Choose an appropriate sampling rate 34
Select the proper location 34
Use the correct wiring 34
SECTION 2 35
PREPARATION FOR USE 35
Unpacking EX10xxA, EX1000A-51C 35
Unpacking RX10xx 35
Unpacking EX1044A/EX1044 35
Installation Location 36

Installation Options 36
Rackmount Installation Option of EX10xxA 36
Rack Ear Installation Option 38
Tabletop Installation Option 39
Installation Option of RX10xx 40
Software Installation 43
Network Configuration 45
Reset button for EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044 45
Network Troubleshooting 46
Restore the EX10xxA's Default Network Settings 46
Determine PCs Network Settings 46
Set the EX10xxA to Auto IP 47
Set the EX10xxA to Static IP 48
Restore the Host PCs Network Settings 48
Using Multiple Network Cards 49
Time Configuration 51
Input Connections / Wiring 51
Thermocouple Connections 51
Voltage Connections 52
RX10xx Connections 54
SECTION 3 57
BASIC OPERATION 57
Introduction 57
Engineering Unit (EU) Conversion 57
Linear Correction 58
Voltage Measurement Ranges 58
Hardware Filter 58
Measurement Range / Input Protection 58
Cold Junction Compensation (CJC) 59
Temperature Units 60
Sampling Rate / Noise Performance 60
Scan List Configuration 61
Scan List Timing 61
Self-calibration 62
Dynamic Calibration in RX1032 63
Open Transducer Detection / Limits 63
Digital I/O and DIO Limit Events 63
LXI Trigger Bus 65
Locking 67
Triggering 67
Data Format 67
Acquiring Data 67
Retrieving Data 68
User-defined Conversions 69
SECTION 4 70
TRIGGERING 70
Overview 70
Events 71
Maximizing Measurement Performance 72
Maximum Trigger Rate 73
SECTION 5 74
WEB PAGE OPERATION 74
Introduction 74
Common Issues 75
Multiple Network Ports 76
Firewall 76

General Web Page Operation 76
Trigger Menu 78
Configuration 78
Initialize 79
Abort 79
Software Arm 79
Software Trigger 79
LAN Events Menu 79
Configuration 79
Event Log 80
Status 81
Scan List Menu 81
Configuration 81
Filters 82
Ranges 82
Data Menu 82
Get FIFO Count 82
Retrieve Data 82
Continuous Polling 82
FIFO Configuration 82
IO Menu 83
Digital IO 83
Trigger Bus 83
Limits Menu 84
Limits 84
DIO Limit Events 84
Device Menu 86
Self-calibration 86
Dynamic Calibration in RX1032 87
Get Calibration Files 87
Network Configuration 88
Device Identify 89
Time Configuration 89
LXI Synchronization 91
Reset Device 92
Hard Reboot 92
Upgrade Firmware 92
Locking Menu 93
Lock 93
Unlock 93
Break Lock 93
Advanced Menu 93
User Conversions 93
User CJC Temp 93
Configuration Examples 94
Example #1 94
Example #2 95

SECTION 6 96

PROGRAMMING 96
Programming Sequence 96
Default Settings 97
EX10xxA Backward Compatibility 98
General Commands / Queries 98
Initializing the device 98
Resetting the device 99
Performing Self-Calibration 99
Dynamic Calibration in RX1032 100

Acquiring a Lock 100
Configure the Scan List 101
Configure the EU Conversions 101
Configure the Range 102
Configure the Advanced Conversion Options 102
Employing an external cold junction 102
Employing custom thermocouple conversions 102
Configure the Voltage Measurement Channels 103
Configure the Filter Frequencies 103
Configure the FIFO 104
Configure the Limit System 104
Configure the Digital I/O System 105
Configure DIO Limit Events 107
Configure the Trigger Model 108
Configure the ARM event system 108
Configure the TRIG event system 110
Configure general trigger model parameters 111
Resetting the trigger model configuration 112
Initiate the Trigger Model 112
Configure the LXI Trigger Bus 112
Trigger Event 114
Retrieving Data (Read FIFO and Streaming Data) 114
Read FIFO 114
Asynchronous Streaming Data 117
Example Program 123
SECTION 7 127
FUNCTION CALLS 127
Introduction 127
Alphabetical Function Set 127
Sample Function Definition 131
Function_Name 131
vtex10xxA_abort 133
vtex10xxA_append_scanlist 134
vtex10xxA_break_lock 135
vtex10xxA_check_lock 136
vtex10xxA_clear_lan_eventlog 137
vtex10xxA_close 138
vtex10xxA_disable_streaming_data 139
vtex10xxA_enable_streaming_data 140
vtex10xxA_enable_streaming_dataEx 142
vtex10xxA_error_message 144
vtex10xxA_error_query 145
vtex10xxA_get_accum_limit_status 146
vtex10xxA_get_alarm 147
vtex10xxA_get_alarm_enable 148
vtex10xxA_get_arm_count 149
vtex10xxA_get_arm_delay 150
vtex10xxA_get_arm_infinite 151
vtex10xxA_get_arm_lan_eventID 152
vtex10xxA_get_arm_lan_filter 153
vtex10xxA_get_arm_source 154
vtex10xxA_get_arm_sourceEx 155
vtex10xxA_get_calibration_file 156
vtex10xxA_get_calibration_running 157
vtex10xxA_get_channel_conversion 158
vtex10xxA_get_channel_count 159
vtex10xxA_get_channel_range 160

vtex10xxA_get_channel_type 161
vtex10xxA_get_dio_input 162
vtex10xxA_get_dio_limit_event 163
vtex10xxA_get_dio_limit_event_invert 164
vtex10xxA_get_dio_limit_event_latch 165
vtex10xxA_get_dio_output 166
vtex10xxA_get_dio_output_enable 167
vtex10xxA_get_fifo_config 168
vtex10xxA_get_fifo_count 169
vtex10xxA_get_filt_freq 170
vtex10xxA_get_init_cont 171
vtex10xxA_get_lan_event_domain 172
vtex10xxA_get_lan_event_source_state 173
vtex10xxA_get_lan_eventlog_count 174
vtex10xxA_get_lan_eventlog_enabled 175
vtex10xxA_get_lan_eventlog_overflowmode 176
vtex10xxA_get_limit_set0 177
vtex10xxA_get_limit_set0_manual 178
vtex10xxA_get_limit_set1 179
vtex10xxA_get_linear_correction 180
vtex10xxA_get_model 181
vtex10xxA_get_OTD_enable 182
vtex10xxA_get_ptp_info 183
vtex10xxA_get_scanlist 184
vtex10xxA_get_serialNumber 185
vtex10xxA_get_shield_gnd_config 186
vtex10xxA_get_system_time 187
vtex10xxA_get_trig_lan_eventID 188
vtex10xxA_get_trig_lan_filter 189
vtex10xxA_get_trigger_count 190
vtex10xxA_get_trigger_delay 191
vtex10xxA_get_trigger_infinite 192
vtex10xxA_get_trigger_source 193
vtex10xxA_get_trigger_sourceEx 194
vtex10xxA_get_trigger_timer 195
vtex10xxA_get_user_cjc_enable 196
vtex10xxA_get_user_cjc_temp 197
vtex10xxA_get_user_conversion 198
vtex10xxA_get_vtb_input 199
vtex10xxA_get_vtb_output 200
vtex10xxA_get_vtb_output_enable 201
vtex10xxA_get_vtb_wiredor 202
vtex10xxA_get_vtb_wiredor_bias 203
vtex10xxA_init 204
vtex10xxA_init_imm 205
vtex10xxA_lock 206
vtex10xxA_pop_logged_LAN_event 207
vtex10xxA_read_fifo 208
vtex10xxA_read_fifoEx 209
vtex10xxA_reset 210
vtex10xxA_reset_fifo 211
vtex10xxA_reset_trigger_arm 212
vtex10xxA_revision_query 213
vtex10xxA_self_cal_clear 214
vtex10xxA_self_cal_clear_stored 215
vtex10xxA_self_cal_get_status 216
vtex10xxA_self_cal_init 217

vtex10xxA_self_cal_is_stored 218
vtex10xxA_self_cal_load 219
vtex10xxA_self_cal_store 220
vtex10xxA_self_test 221
vtex10xxA_self_test_init 222
vtex10xxA_set_alarm 223
vtex10xxA_set_alarm_enable 224
vtex10xxA_self_test_init 225
vtex10xxA_set_arm_count 226
vtex10xxA_set_arm_delay 227
vtex10xxA_set_arm_infinite 228
vtex10xxA_set_arm_lan_eventID 229
vtex10xxA_set_arm_lan_filter 230
vtex10xxA_set_arm_source 231
vtex10xxA_set_arm_sourceEx 232
vtex10xxA_set_channel_conversion 233
vtex10xxA_set_channel_range 234
vtex10xxA_set_communication_timeout 235
vtex10xxA_set_dio_limit_event 236
vtex10xxA_set_dio_limit_event_invert 237
vtex10xxA_set_dio_limit_event_latch 238
vtex10xxA_set_dio_output 239
vtex10xxA_set_dio_output_enable 240
vtex10xxA_set_dio_pulse 241
vtex10xxA_set_fifo_config 242
vtex10xxA_set_filt_freq 243
vtex10xxA_set_init_cont 244
vtex10xxA_set_lan_event_domain 245
vtex10xxA_set_lan_eventlog_enabled 246
vtex10xxA_set_lan_eventlog_overflowmode 247
vtex10xxA_set_limit_set0 248
vtex10xxA_set_limit_set0_manual 249
vtex10xxA_set_limit_set1 250
vtex10xxA_set_linear_correction 251
vtex10xxA_set_OTD_enable 252
vtex10xxA_set_scanlist 253
vtex10xxA_set_trig_lan_eventID 254
vtex10xxA_set_trig_lan_filter 255
vtex10xxA_set_trig_source_timer 256
vtex10xxA_set_trigger_count 257
vtex10xxA_set_trigger_delay 258
vtex10xxA_set_trigger_infinite 259
vtex10xxA_set_trigger_source 260
vtex10xxA_set_trigger_sourceEx 261
vtex10xxA_set_trigger_timer 262
vtex10xxA_set_user_cjc_enable 263
vtex10xxA_set_user_cjc_temp 264
vtex10xxA_set_user_conversion 265
vtex10xxA_set_vtb_output 266
vtex10xxA_set_vtb_output_enable 267
vtex10xxA_set_vtb_pulse 268
vtex10xxA_set_vtb_wiredor 269
vtex10xxA_set_vtb_wiredor_bias 270
vtex10xxA_soft_arm 271
vtex10xxA_soft_trigger 272
vtex10xxA_unlock 273
vtex10xxA_get_shield_gnd_config 274

vtex10xxA_set_communication_timeout....275
vtex10xxA _get_utc_offset....276
vtex10xxA_self_cal_init_ex....277
Error Messages....278
SECTION 8....280
THEORY OF OPERATION....280
Introduction....280
Voltage / Thermocouple Input Options....280
Thermocouple Connector Inputs....280
D-sub Connector Inputs....281
Signal Conditioning Circuitry....281
Cold Junction Compensation....282
Calibration....282
Thermocouple Calculations....282
SECTION 9....283
EX10SC SIGNAL CONDITIONING....283
Overview....283
Connecting Power to the EX10SC....283
Connecting to an EX10xxA to the EX10SC....284
Connector Pin/Signal Assignment....284
Connecting to the EX10SC to the DUT....285
Changing Signal Conditioning Modules....286
Jumper Settings....286
Shunt Resistors....287
Software Interface....288
EX10SC General Specifications....292
EX10SC Module Specifications....293
SECTION 10....303
EX10XXA/EX10XXA-51C/RX10XX/EX1044A/EX1044CALIBRATION....303
Introduction....303
Required Resources....303
Equipment Needed....303
Software Needed....303
Setting Up the Instruments....304
Connecting to the EX10xxA/RX10xx/EX1044A/EX1044....305
Login into SFP....306
Time Configuration....306
Performing Calibration....307
Monitoring Calibration Progress....308
SECTION 11....309
ONBOARD MEMORY....309
Onboard Memory and Clearing Procedure....309
SECTION 12....310
QUICK REFERENCE GUIDE FOR RX1032....310
SECTION 13....315
QUICK REFERENCE GUIDE FOR EX1044A/EX1044....315
INDEX....318

ABOUT AMETEK

AMETEK Programmable Power, Inc., a Division of AMETEK, Inc., is a global leader in the design and manufacture of precision, programmable power supplies for R&D, test and measurement, process control, power bus simulation and power conditioning applications across diverse industrial segments. From bench top supplies to rackmounted industrial power subsystems, AMETEK Programmable Power is the proud manufacturer of Elgar, Sorensen, California Instruments, Amrel brand power supplies. Also VTI Instruments brand which delivers precision modular instrumentation and systems for electronic signal distribution, acquisition, and monitoring, used in the world's most demanding test applications.
AMETEK, Inc. is a leading global manufacturer of electronic instruments and electromechanical devices with annualized sales of $4 billion. The Company has over 15,000 colleagues working at nearly 150 manufacturing facilities and nearly 150 sales and service centers in the United States and 30 other countries around the world.

TRADEMARKS

AMETEK is a registered trademark of AMETEK, Inc. VTI Instruments is a trademark owned by AMETEK, Inc. Other trademarks, registered trademarks, and product names are the property of their respective owners and are used herein for identification purposes only.

Notice of Copyright

EX14xx Series User Manual © 2018 AMETEK Programmable Power, Inc. All rights reserved.

EXCLUSION FOR DOCUMENTATION

UNLESS SPECIFICALLY AGREED TO IN WRITING, AMETEK PROGRAMMABLE POWER, INC. (“AMETEK”):

(a) MAKES NO WARRANTY AS TO THE ACCURACY, SUFFICIENCY OR SUITABILITY OF ANY TECHNICAL OR OTHER INFORMATION PROVIDED IN ITS MANUALS OR OTHER DOCUMENTATION.

(b) ASSUMES NO RESPONSIBILITY OR LIABILITY FOR LOSSES, DAMAGES, COSTS OR EXPENSES, WHETHER SPECIAL, DIRECT, INDIRECT, CONSEQUENTIAL OR INCIDENTAL, WHICH MIGHT ARISE OUT OF THE USE OF SUCH INFORMATION. THE USE OF ANY SUCH INFORMATION WILL BE ENTIRELY AT THE USER’S RISK, AND

(c) GIVES NOTIFICATION THAT, IF THIS MANUAL IS IN ANY LANGUAGE OTHER THAN ENGLISH, ALTHOUGH STEPS HAVE BEEN TAKEN TO MAINTAIN THE ACCURACY OF THE TRANSLATION, THE ACCURACY CANNOT BE GUARANTEED. APPROVED AMETEK CONTENT IS WITHIN THE ENGLISH LANGUAGE VERSION, WHICH IS POSTED AT WWW.PROGRAMMABLEPOWER.COM.

CONTACT INFORMATION

Telephone: +1 800 733 5427 (toll free in North America)
+1 858 450 0085 (direct)
Fax: +1 858 458 0267
Email: sales.ppd@ametek.com
service.ppd@ametek.com
Web: www.powerandtest.com

CERTIFICATION

VTI Instruments (VTI) certifies that this product met its published specifications at the time of shipment from the factory. VTI further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology (formerly National Bureau of Standards), to the extent allowed by that organization's calibration facility, and to the calibration facilities of other International Standards Organization members.

WARRANTY TERMS

The product referred to herein is warranted against defects in material and workmanship for a period of one year from the receipt date of the product at customer's facility.
AMETEK Programmable Power, Inc. ("AMETEK"), provides this written warranty covering the Product stated above, and if the Buyer discovers and notifies AMETEK in writing of any defect in material or workmanship within the applicable warranty period stated above, then AMETEK may, at its option: repair or replace the Product; or issue a credit note for the defective Product; or provide the Buyer with replacement parts for the Product.
The Buyer will, at its expense, return the defective Product or parts thereof to AMETEK in accordance with the return procedure specified below. AMETEK will, at its expense, deliver the repaired or replaced Product or parts to the Buyer. Any warranty of AMETEK will not apply if the Buyer is in default under the Purchase Order Agreement or where the Product, or any part thereof, is as follows:

- damaged by misuse, accident, negligence or failure to maintain the same as specified or required by AMETEK;
- damaged by modifications, alterations or attachments thereto which are not authorized by AMETEK;
- installed or operated contrary to the instructions of AMETEK;
- opened, modified, or disassembled in any way without AMETEK's consent;
- used in combination with items, articles or materials not authorized by AMETEK.

The Buyer may not assert any claim that the Products are not in conformity with any warranty until the Buyer has made all payments to AMETEK provided for in the Purchase Order Agreement.

PRODUCT RETURN PROCEDURE

Request a Return Material Authorization (RMA) number from the repair facility (**must be done in the country in which it was purchased**):

- **In the USA**, contact the AMETEK Customer Service Department prior to the return of the product to AMETEK for repair:

  Telephone: 800-733-5427, ext. 2295 or ext. 2463 (toll free North America)
  858-450-0085, ext. 2295 or ext. 2463 (direct)
- **Outside the United States**, contact the nearest Authorized Service Center (ASC). A full listing can be found either through your local distributor, or on our website, https://www.powerandtest.com/service-and-support/rma/rma-request-form.

When requesting an RMA, have the following information ready:

- Model number
- Serial number
- Description of the problem

NOTE: Unauthorized returns will not be accepted and will be returned at the shipper's expense.
NOTE: A returned product found upon inspection by AMETEK to be in specification is subject to an evaluation fee and applicable freight charges.

DECLARATION OF CONFORMITY

Declaration of Conformity According to EN ISO/IEC 17050-1:2004

MANUFACTURER'S NAME	VTI Instruments Corp.
MANUFACTURER'S ADDRESS	2031 Main Street Irvine, California 92614-6509
PRODUCT NAME	Precision Voltage/Temperature Measurement Instrument
MODEL NUMBER(S) EX10SC	EX1000A, EX1000A-TC, EX1016A, EX1032A, EX1048A,
PRODUCT OPTIONS	All
PRODUCT CONFIGURATIONS	All

VTI Instruments declares that the aforementioned product conforms to the requirements of the Low Voltage directive (European Council directive 2014/35/EU, dated 22 July 1993) and the Electromagnetic Compatibility directive (European Council directive 2014/30/EU; generally referred to as the EMC directive). In substantiation, the products were tested and/or evaluated to the standards shown below:

SAFETY	EN61010-1:2010
EMC	EN61326-1:2013 EN55011 Class A Group 1 EN61000-4-2 EN61000-4-3 EN61000-4-4 EN61000-4-5 EN61000-4-6 EN61000-4-8 EN61000-4-11 CISPR 22

June 2016

CE

Steve Mauga, QA Manager

DECLARATION OF CONFORMITY

Declaration of Conformity According to EN ISO/IEC 17050-1:2004

MANUFACTURER'S NAME	VTI Instruments Corp.
MANUFACTURER'S ADDRESS	2031 Main Street Irvine, California 92614-6509
PRODUCT NAME	Rugged Precision Temperature & Voltage Measurement Instrument
MODEL NUMBER(S)	RX1032
PRODUCT OPTIONS	All
PRODUCT CONFIGURATIONS	All

VTI Instruments declares that the aforementioned product conforms to the requirements of the Low Voltage directive (European Council directive 2014/35/EU, dated 22 July 1993) and the Electromagnetic Compatibility directive (European Council directive 2014/30/EU; generally referred to as the EMC directive). In substantiation, the products were tested and/or evaluated to the standards shown below:

EMC	EN61326-1:2013 EN55011 Class A Group 1 EN61000-4-2 EN61000-4-3 EN61000-4-4 EN61000-4-5 EN61000-4-6 EN61000-4-8 EN61000-4-11 CISPR 11
SHOCK AND VIBRATION	MIL-PRF-28800F RH: Class 2 (3.8.2.3 & 4.5.5.1) Vibration: Class 2 (3.8.4.1 & 4.5.5.3.1) Shock (Functional) Class 1 (3.8.5.1 & 4.5.5.4.1)
INGRESS PROTECTION (IP 65)	EN60529:1992+A2:2013

October 2016

CE

Steve Mauga, QA Manager

GENERAL SAFETY INSTRUCTIONS

Review the following safety precautions to avoid bodily injury and/or damage to the product. These precautions must be observed during all phases of operation or service of this product. Failure to comply with these precautions, or with specific warnings elsewhere in this manual, violates safety standards of design, manufacture, and intended use of the product. Note that this product contains no user serviceable parts or spare parts.

Service should only be performed by qualified personnel. Disconnect all power before servicing.

TERMS AND SYMBOLS

These terms may appear in this manual:

WARNING Indicates that a procedure or condition may cause bodily injury or death.

CAUTION Indicates that a procedure or condition could possibly cause damage to equipment or loss of data.

These symbols may appear on the product:



ATTENTION - Important safety instructions



Frame or chassis ground



Indicates that the product was manufactured after August 13, 2005. This mark is placed in accordance with *EN 50419, Marking of electrical and electronic equipment in accordance with Article 11(2) of Directive 2002/96/EC (WEEE)*. End-of-life product can be returned to VTI by obtaining an RMA number. Fees for take-back and recycling will apply if not prohibited by national law.

WARNINGS

Follow these precautions to avoid injury or damage to the product:

Use Proper Power Cord

The power cable provided with this instrument meets the required regulatory and statutory safety standards as indicated by this product's declaration of conformity. VTI recommends that the power cord provided be used with the instrument that it is provided with. If a different power cord is must to be used, however, it is the responsibility of the user to select a power cord that meets any and all regulatory and statutory requirements for their industry and country.

Do not use the power cord supplied with this instrument with any other equipment.

Use Proper Power Source

To avoid electrical overload, electric shock, or fire hazard, do not use a power source that applies other than the specified voltage.

The mains outlet that is used to power the equipment must be within 3 meters of the device and shall be easily accessible.

WARNINGS (CONT.)

Avoid Electric Shock

To avoid electric shock or fire hazard, do not operate this product with the covers removed. Do not connect or disconnect any cable, probes, test leads, etc. while they are connected to a voltage source. Remove all power and unplug unit before performing any service. ***Service should only be performed by qualified personnel.***

Ground the Product

This product is grounded through the grounding conductor of the power cord. To avoid electric shock, the grounding conductor must be connected to earth ground.

Operating Conditions

To avoid injury, electric shock or fire hazard:

- Do not operate in wet or damp conditions.
- Do not operate in an explosive atmosphere.
- Operate or store only in specified temperature range.
- Provide proper clearance for product ventilation to prevent overheating.
- DO NOT operate if any damage to this product is suspected. ***Product should be inspected or serviced only by qualified personnel.***



Improper Use

The operator of this instrument is advised that if the equipment is used in a manner not specified in this manual, the protection provided by the equipment may be impaired.
Conformity is checked by inspection.

SECTION 1

INTRODUCTION

OVERVIEW

In general, EX10xxA family of products provides a total of 48-channels, with an exception of RX10xx products which has 32-channels, of high-performance thermocouple and/or voltage measurement in a single instrument. This family includes the EX1000A, EX1000A-TC, EX1016A, EX1032A, EX1048A, EX1000A-51C and RX1032. The last two digits of the model name indicate the number of thermocouple channels/voltage channels available on the model. For example, the EX1016A has sixteen thermocouple channels and thirty-two voltage channels. Its combination of measurement performance and integrity, configuration flexibility, package density, and network connectivity make it the most powerful, yet easy-to-use, instrument of its kind. The EX10xxA is a complete, self-contained temperature measurement system that communicates over Ethernet. Unlike other data acquisition offerings in its class, the EX10xxA offers a tightly integrated solution that frees the user from the complexity of marrying terminal blocks, signal conditioning cards, digitizer, power supply, and chassis together.

EX1000A-51C is the enhanced version of the EX10xxA family of products, has 51 Voltage Channels and has additional features such as additional sensors to measure internal box temperature and additional API's provided to read UTC offset & warmup time.

RX10xx is the rugged version of the EX10xxA family of products, has 32 Channels and most importantly provides the user with the flexibility to place this instrument even in extreme conditions and directly in the test cells because of its rugged build.

EX1044A is the customized version of the EX10xxA family of products, has 44 TC Channels and 2 channels for voltage and 2 channels for current whereas EX1044 has only 44 TC Channels with 5B module slots provision (user can select any 5B modules). This provides thermal labs customers the ability to place instrumentation directly in the thermal test labs. The EX10xx product has already demonstrated some of the highest accuracy in the market for Thermocouple measurements and the EX1044A/EX1044 will continue to provide as good or better accuracy for gas turbine test cell users trying to improve engine efficiency by better understanding temperature effects on the DUT.

The EX10xxA also provides a level of measurement integrity and channel independence that far exceeds the typical data acquisition system. Absent are caveats about scanning speed, channel order, and overload effects. In the EX10xxA, channels have no influence on each other, regardless of scanning speed or channel state. Excellent common mode rejection performance provides immunity from not only power line interference, but also high frequency noise. Moreover, it aids in maintaining overall system integrity by offering a high-performance open transducer detection system.

The EX10xxA provides a high level of configuration flexibility as well. Each channel can be configured independently with regards to measurement function, hardware filter setting, and limit values. In addition to measuring all standard thermocouples, the EX10xxA can be programmed with user-defined thermocouple polynomial equations or be used as a low-noise voltmeter. Scanning speed is programmable up to a maximum of 1 kSa/s, independent of the number of channels being scanned. This combination of filtering and scanning speed provides the EX10xxA with the low noise

performance required for sensitive applications as well as the speed necessary to measure fast, fine-gauge thermocouples.

For highest accuracy and stability, the EX10xxA provides an embedded isothermal input section that is monitored by up to twelve precision thermistors (depending on model), one for every four thermocouple channels. Moreover, it features an internal calibration source that can be used to self-calibrate the unit upon command. This extends the unit's high accuracy over a wide ambient temperature range. Thermocouple accuracy is detailed in Table 1-2.

The EX10xxA is as easy-to-use as it is powerful. An integrated web page provides a convenient way to instantly verify communications and instrument functionality, while industry standard VXI *plug & play* and IVI drivers provide a familiar application programming interface to reduce integration and program development time.

FEATURES

Combined Thermocouple and Analog Inputs

Easily combine thermocouple and analog signals on these versatile instruments with sampling rates up to 1 kSa/s (kilo samples per second) per channel. Each input incorporates an independent signal conditioning path with software selectable filters for maximum flexibility. Complete channel independence ensures data integrity regardless of sample speed or input overload conditions.

End-to-end Self-calibration

Complete end-to-end self-calibration is provided for each signal path on a programmable basis. A highly accurate calibration source provides reference signals that are applied prior to analog filtering and gain circuits to compensate for drift, aging, or temperature variations. Self-calibration is simple and quick, and can be performed as often as desired.

Scalable for Synchronized High-Speed, High-Channel Count

With LXI™ Class A compliant features like a built-in Trigger Bus™ hardware trigger subsystem, the EX10xxA supports easy integration and synchronization of multiple devices, including existing VXI bus instrumentation.

Open Thermocouple Detection (OTD)

While the integration of the EX10xxA removes many of the reliability and connectivity problems typically faced by system designers, they still must contend with the reliability of the sensor connections. Fortunately, the EX10xxA aids a great deal in that regard as well. Each thermocouple input channel generates a very small current (±7.5 nA/channel) which drives the input to an open-circuit state if the transducer is disconnected. Implemented in this way, open transducer detection is continuous and requires no discrete function call on the part of the user to be activated, offering more protection than systems that only check on command, as a broken sensor can occur at any time during testing, not just at installation. In a typical application, the OTD has no effect on the measurement. However, many thermocouple simulators have a much higher impedance than a thermocouple so an offset may be introduced into the reading when using a simulator. In general, the OTD should only be active on channels that are being used with actual thermocouples.

The EX10xxA additionally offers a front panel OTD LED for each channel that illuminates upon the recognition of a fault condition. This provides for quick and easy problem channel identification. Moreover, to ensure that even intermittent problems are identified, the fault recognition is a latching mechanism, retaining the information of the current acquisition sequence until a new acquisition is initiated. Note that voltage channels do not include internal CJC thermistors, but that each voltage D-sub includes connections for the connection of user-supplied thermistors.

Note: OTD functionality will not be applicable for EX1000A-51C. RX1032 doesn't have option to provide OTD LED indication in a fault condition due to its rugged design.

Cold Junction Compensation (CJC)

For highest accuracy and stability, the EX1000A-TC, EX1016A, EX1032A, EX1048A, EX1044, EX1044A and RX1032 provide embedded isothermal input sections that are monitored by precision thermistors (based on Model#, the number of thermistors can vary), one for every four thermocouple channels. (Note, the EX1000A may utilize external thermistors as described in the *Voltage Connections* section). To ensure that the CJC information is current and time correlated with the input channels, the CJC channels are measured with every scan, providing a maximum time separation of less than 4 ms between the measurement of an input channel and its associated CJC measurement. For those users that prefer to employ an external cold junction, the EX10xxA also allows for the programming input of up to forty-eight unique external cold junction temperatures, one for every input channel. Moreover, the use of internal and external CJC inputs can be mixed throughout the unit on a per channel basis.

Simplified Installation, Setup and Control

Full LXI Class A compliance makes the EX10xxA of instruments ideal for distributed measurements throughout a facility – reducing cabling and installation expense. Connect directly to an Ethernet network using industry standard Ethernet cable and connections. Experience how *plug&play* capability simplifies installation and setup.

An onboard, web-accessible user interface allows you to instantly verify communications and instrument functionality, while IVI and VXI *plug&play* drivers provide a familiar application programming interface to further reduce integration and program development time. For comprehensive, programming-free data recording setup, management and viewing, use EX10xxA instruments with one of VTI's full-featured, turnkey software solutions: EXLab or VTICODA.

Channel Independence

Each of the EX10xxA's 48 differential input channels is an independent signal conditioning path, complete with amplification, programmable hardware filtering, and continuous open transducer detection. This independence frees the user from the problem of channel-to-channel crosstalk that is pervasive in most multi-channel data acquisition systems. In the EX10xxA, channels have no influence on each other, regardless of scanning speed or channel state. Specifically, open or significantly overloaded channels do not affect the measurement results of any other channels.

Measurement Range

The measurement range of the EX10xxA in terms of temperature is a function of its input voltage range and the capabilities of the thermocouple sensors themselves. Specifically, the measurement range of the EX10xxA for the standard thermocouple types is the following:

Type	Min Temp	Max Temp
J	-200 °C (-328 °F)	1200 °C (2192 °F)
K	-200 °C (-328 °F)	1372 °C (2502 °F)
T	-200 °C (-328 °F)	400 °C (752 °F)
E	-200 °C (-328 °F)	900 °C (1652 °F)
S	-50 °C (-58 °F)	1768 °C (3214 °F)
R	-50 °C (-58 °F)	1768 °C (3214 °F)
B	250 °C (482 °F)	1820 °C (3308 °F)
N	-200 °C (-328 °F)	1300 °C (2372 °F)

TABLE 1-1: EX10XXA MEASUREMENT RANGE

In terms of voltage, the thermocouple connectors may only accept an input voltage of ±67 mV when performing temperature conversion. When in voltage mode, these connectors can accept either ±0.067 V or ±0.1 V inputs. The voltage connectors may use either ±0.01 V, ±0.067 V, ±0.1 V,

±1.0 V, or ±10.0 V inputs for both EU temperature conversion or voltage measurements. When temperature is selected, the instrument automatically switches the selected channel(s) to the ±67 mV range. The EX1000A-TC is a special case where thermocouple inputs are used for voltage measurement and may utilize all input ranges available to the voltage connectors. Table 1- provides an overview of each models capabilities.

For maximum flexibility, each channel can be independently configured with regards to its thermocouple conversion. Moreover, non-standard thermocouples are accommodated through the input of user-defined thermocouple polynomial coefficients.

Hardware Filter

Each EX10xxA input channel can be individually configured with one of six low-pass hardware filters. Five of these filters are 2-pole Bessel filters with frequencies of 4 Hz, 15 Hz, 40 Hz, 100 Hz and 500 Hz. The sixth filter is a single-pole Butterworth filter with a frequency of 1 kHz, providing a simple anti-alias filter when the 1 kSa/s range is used. These filters allow the EX10xxA to cover a diverse range of applications, even within the same unit. Suitable for most applications, the 4 Hz setting provides the lowest noise floor and exceptional common mode rejection. For higher speed applications, the 1 kHz filter will pass the output from even the fastest fine-gauge thermocouples with little distortion.

Input Connectors

The EX1016A, EX1032A, EX1048A, EX1044 and EX1044A employ uncompensated (Cu-Cu) mini-thermocouple female jacks as an input connector. This connector provides a solid, reliable connection that is also easily changeable. Since it is not thermocouple-type specific, different thermocouple types can be mixed throughout the unit without hardware modification. The EX1000A, EX1016A and EX1032A also employ a standard 50-pin D-sub connector for the voltage input channels. All the RX10xx channels are individually routed via the screw terminals which is placed just below the instrument top cover and the cable bunch is brought out through the cable gland.

Sampling Rate

The EX10xxA can be configured for a sampling rate up to a maximum of 1 kSa/s, regardless of the number of channels included in the scan list. When the requested sampling rate is significantly less than 1 kSa/s, however, the EX10xxA automatically takes and averages multiple samples. This offers improved noise performance, while maintaining the requested data output rate.

Digital I/O and Limits

The EX10xxA provides two unique sets of programmable limits that are used for open transducer detection as well as general purpose input channel monitoring. These limits are programmable on a per channel basis and are evaluated with each completed scan. The output of limit evaluations is presented in three forms. The operation of the front panel LEDs is tied to the upper and lower limit values of one limit set. The operation of the digital I/O port can be optionally linked to any combination of the upper and lower limit values of either or both limit sets. Finally, the limit condition information is accessible through the instrument driver.

The EX10xxA features an 8-channel digital I/O port on the rear panel of the instrument, except RX10xx which has 2-channel digital I/O routed through the 13 pin circular connector (P1). This port can be used as an arm/trigger source, for presentation of limit evaluation information, and as a general purpose output device. As a general purpose output device, each DIO channel can be independently programmed with regards to its output functionality and its static level to assume when enabled as an output. For expanded and more automated operation, each DIO channel can be independently linked to one or multiple limit conditions on one or more input channels.

LXI Trigger Bus

The EX10xxA features an 8-channel LXI (*LAN eXtensions for Instrumentation*) trigger bus on the rear panel of the instrument. This differential-pair LVDS (Low Voltage Differential Signal) bus consists of two identical ports connected in parallel. The primary use of the trigger bus is the transmission of high-speed signals for multiple-unit triggering and synchronization.

Note: RX1032, EX1044 and EX1044A doesn't have LXI trigger bus.

Triggering

The EX10xxA supports a full function trigger model with a separate arm source and trigger source event structure. Trigger and arm source events can be independently programmed from a variety of sources including Immediate, Timer, Digital I/O, LAN events, the LXI Trigger Bus, and the LXI alarm.

EX10XXA SPECIFICATIONS

GENERAL SPECIFICATIONS	
NUMBER OF CHANNELS	
EX10XXA	48 differential inputs
EX10XXA-51C	51 differential inputs
RX10XX	32 differential inputs
EX1044A	44 differential inputs (TC inputs), 2 Voltage and 2 Current
EX1044	44 differential inputs (TC inputs), 4ch 5B Module options
CHANNEL TYPES	
Thermocouple and voltage	J, K, T, E, S, R, B, N (EX1000A-TC, EX1016A, EX1032A, EX1048A, RX1032, EX1044, EX1044A) mV (RX1032), V (EX1000A, EX1000A-TC, EX1016A, EX1032A, EX1000A-51C, EX1044, EX1044A, RX1000) mA, A (EX1044)
SAMPLING RATE	
	1000 Sa/s per channel maximum
TEMPERATURE RESOLUTION	
	0.1 °C
TEMPERATURE ACCURACY	
	See Table 1-2 below
TEMPERATURE NOISE	
Peak-to-peak	0.08 °C typical (J, K, T, E)
PROGRAMMABLE FILTERS	
Bessel (2 pole)	4 Hz, 15 Hz, 40 Hz, 100 Hz, and 500 Hz (-3 dB cutoff frequency)
Butterworth (1 pole)	1000 Hz (-3 dB cutoff frequency)
VOLTAGE INPUT RANGE	
Voltage input channels	±0.01 V, ±0.067 V, ±0.1 V, ±1.0 V, ±10.0 V
Thermocouple input channels	±0.067 V for temperature measurement, ±0.067 V and ±0.1 V for voltage measurement.
INPUT POWER	
Input voltage/frequency	90 V ac – 264 V ac*, 50 Hz/60 Hz (nominal ac)
Power	47 VA
*Note, fluctuations for mains voltage to the power supply not exceeding 10% of the nominal voltage.	
VOLTAGE RESOLUTION	
±10.0 V	300 µV
±1.0	30 µV
±0.1 V	3.0 µV
±0.067 V	2.0 µV
±0.01 V	0.3 µV
VOLTAGE ACCURACY	**± (% of reading + offset)**
±10.0 V	± (0.025% + 500 µV) with self-cal*, ± (0.05% + 1 mV) without self-cal*
±1.0 V	± (0.025% + 50 µV) with self-cal*, ± (0.05% + 100 µV) without self-cal*
±0.1 V	± (0.025% + 10 µV) with self-cal*, ± (0.05% + 20 µV) without self-cal*
±0.067 V	± (0.025% + 10 µV) with self-cal*, ± (0.05% + 20 µV) without self-cal*

GENERAL SPECIFICATIONS

±0.01 V ***Note, self-cal is not applicable for RX1032**	± (0.050% + 10 µV) with self-cal*, ± (0.10% + 20 µV) without self-cal*

VOLTAGE OFFSET STABILITY	
±10.0 V	± 20 µV/°C typical
±1.0 V	± 10 µV/°C typical
±0.1 V	± 5 µV/°C typical
±0.067 V	± 2 µV/°C typical
±0.01 V	± 2 µV/°C typical

VOLTAGE GAIN STABILITY	
Voltage input channels (all ranges) and thermocouple input channels ***Note, self-cal is not applicable for RX1032**	± 25 ppm/°C without self-cal* (typical) ± 5 ppm/°C with self-cal* at any operating temperature (typical)

INPUT IMPEDANCE	
	40 MΩ differential

INPUT BIAS CURRENT	
	5 nA typical

COMMON MODE INPUT RANGE	
	±10 V

COMMON MODE REJECTION RATIO (CMRR)	
4 Hz filter	dc: 100 dB minimum; (50/60) Hz: 140 dB typical, 120 dB minimum
1 kHz filter	dc: 100 dB minimum; (50/60) Hz: 100 dB typical, 80 dB minimum

See Figure 6-2 in the Theory of Operation for more information regarding CMRR.

INPUT PROTECTION	
	±15 V

NETWORK CONNECTION	
	10/100 Base-T

INPUT CONNECTORS	
Thermocouple	
EX10xxA	Cu-Cu mini-TC jack
RX10xx	Screw terminals (three terminals for each channels "S, +, -") M3 X 6MM (PAN HEAD PHILLIPS)
Analog voltage	
EX10xxA	50-pin female D-sub
EX1000A-51C	50-pin female D-sub
EX1000A-TC	Cu-Cu mini-TC jack

OPERATING TEMPERATURE (FOR EX10XX A)	
	0 °C to 50 °C

POWER INPUT (EX10XXA, 51C)	
EX10xxA	(90 – 264) V ac, (50/60) Hz
RX10xx	INPUT VOLTAGE DC, 18-36V DC

DIMENSIONS (EX10XXA/RX10XX)	
EX10xxA	1.75'' H x 17.5'' W x 13.6''D
EX1000A-51C	1.75'' H x 17.5'' W x 13.6''D
RX10xx	4.92" H x 9.05" W x 9.45" D; 125 x 238 x 249 mm (including handles and latches)

WEIGHT (RX10XX)	
RX10xx	5.3 KG; 11 lbs

RELIABILITY (EX10XXA)	
MTBF	> 100,000 hrs (or approximately 11 years)
Methodology	Telcordia (Bellcore) SR-332, Issue 1

LXI SPECIFICATIONS

LXI CLASS COMPLIANCE	
	LXI Class A

CLOCK SPECIFICATIONS	
Clock oscillator accuracy	±50 ppm
Synchronization accuracy	Reports "synchronized" when < ±200 µs of the 1588 master clock
Timestamp	
Accuracy	As good as time synchronization down to 50 ns
Resolution	25 ns

IEEE 1588-BASED TRIGGER TIMING

Alarm	
Trigger time accuracy	As good as time synchronization down to 50 us
Time to trigger delay	50 us
Receive LAN[0-7] Event	
Trigger time accuracy	As good as time synchronization down to 50 us
Time to trigger delay	
Future timestamp	50 us typical
Past/zero timestamp	1 ms maximum
HARDWARE TRIGGER TIMING	
LXI Trigger Bus	
Time to trigger delay	55 us typical
DIO Bus	
Time to trigger delay	57 us typical

ENVIRONMENTAL SPECIFICATIONS FOR EX10XXA, EX1000A-51C

OPERATING LOCATION	
	This chassis should be operated indoors in a controlled environment, protected from exposure to the elements (i.e. direct sunlight, precipitation, wind, etc.).
TEMPERATURE	
Operating	0 °C to +50 °C
Storage	-40 °C to +70 °C
HUMIDITY	
	5% – 95% (non-condensing)
ALTITUDE	
	Up to 3000 m
SHOCK AND VIBRATION	
	Conforms to MIL-PRF-28800F, Paragraphs 4.5.5.3.1 (random vibration test), 4.5.5.3.2 (sinusoidal vibration test), and 4.5.5.4.1 (functional shock test)

ENVIRONMENTAL SPECIFICATIONS FOR RX10XX

TEMPERATURE	
	-40°C TO +71°C (REDUCED ACCURACY), 0°C TO +50°C (FULL ACCURACY)
HUMIDITY	
	IP 65 RATED ENCLOSURE, 5% TO 95%
ALTITUDE	
	UP TO 4600 M
SHOCK AND VIBRATION	CONFORMS TO MIL-PRF-28800F
	RANDOM VIBRATION 30 MIN PER AXIS, 10-500Hz MIL-PRF-28800F CLASS 2 SINUSOIDAL 5 TO 55HZ RESONANCE SEARCH PER MIL-PRF-28800F CLASS 2, EACH AXIS SHOCK 30G/AXIS, 11MS HALF SINE PULSE PER MIL-PRF-28800F CLASS 2
CHASSIS MOUNTING	
	6 X M6 X 10mm LG, CAP FLANGE HD, 10.9 STEEL SCREWS

ENVIRONMENTAL SPECIFICATIONS FOR EX1044A/EX1044

TEMPERATURE	
	-10°C TO +45°C (REDUCED ACCURACY), 0°C TO +50°C (FULL ACCURACY)
HUMIDITY	
	5% – 95% (non-condensing)
ALTITUDE	
	Up to 3000 m
SHOCK AND VIBRATION	
	Conforms to MIL-PRF-28800F, Paragraphs 4.5.5.3.1 (random vibration test), 4.5.5.3.2 (sinusoidal vibration test), and 4.5.5.4.1 (functional shock test)
CHASSIS MOUNTING	
	TABLE TOP MOUNTING

TABLE 1-2: THERMOCOUPLE ACCURACY FOR EX10XXA (TYPICAL)

Type	Min	Max	-100	0	100	300	500	700	900	1100	1400
J	-200	1200	±0.25	±0.20	±0.20	±0.25	±0.30	±0.30	±0.35	±0.45	
K*	-200	1372	±0.25	±0.20	±0.20	±0.20	±0.35	±0.35	±0.45	±0.55	±0.50
T**	-200	400	±0.25	±0.20	±0.20	±0.20	±0.25				
E	-200	900	±0.25	±0.20	±0.20	±0.20	±0.25	±0.30	±0.35		
S	-50	1768		±1.00	±0.75	±0.65	±0.65	±0.65	±0.70	±0.70	±0.75
R	-50	1768		±1.00	±0.75	±0.60	±0.60	±0.60	±0.60	±0.65	±0.70
B	250	1820				±1.65	±1.10	±0.80	±0.70	±0.65	±0.65
N	-200	1300	±0.40	±0.25	±0.25	±0.25	±0.30	±0.35	±0.40	±0.40	

Values in °C

Conditions
- 60 minute warm-up
- Guaranteed maximum limits are two times (2x) the typical values
- 7 days, ±5 °C from last self-calibration***
- 20 °C to 30 °C, 1 year from full calibration
- Exclusive of thermocouple errors
- Exclusive of noise
- Common mode voltage = 0

* 1400 accuracy is for 1372 °C
** 500 accuracy is for 400 °C

Table 1-3: Thermocouple Accuracy for RX1032

	-40 °C to +50 °C	+50 °C to +70 °C	+15 °C to +25 °C
	Type E, J, K, N & T		
Typical	±0.6°C	±0.6°C	±0.4°C
Max	±1.0°C	±1.0°C	±0.6°C
	Type R & S		
Typical	±0.88°C	±2.12°C	±0.48°C
Max	±1.40°C	±3.72°C	±0.80°C
	Type B		
Typical	±1.76°C	±4.24°C	±0.96°C
Max	±2.80°C	±7.44°C	±1.60°C

Note: Typical is based on 2σ & Max is based on 4σ

Conditions
- After 1 hour warmup period and temperature stable for 1 hour after any temperature change.
- Accuracy does not include accuracy of thermocouple or thermocouple wiring.
- Maximum Temperature rate of change between any operating temperature is 0.5 °C/min
- Guaranteed maximum limits are two times (2x) the typical values
- 7 days, ±5 °C from last calibration
- -20 °C to +30 °C, 1 year from full calibration
- Exclusive of thermocouple errors
- Exclusive of noise
- Common mode voltage = 0
- Self calibration is not applicable

Temperature rate of change
- Nominal - 0.33 °C /min
- Max - 1.33 °C /min

TABLE 1-4: EX10XXA/RX10XX CONFIGURATIONS

	EX1000A	EX1016A	EX1032A	EX1048A	EX1000A-TC	EX1000A-51C	RX1032	EX1044A	EX1044
±10 V Range	48 channels	32 channels [1,2,3]	16 channels [1,2,3]		48 channels	51 channels			
±1.0 V Range	48 channels	32 channels [1,2,3]	16 channels [1,2,3]		48 channels	51 channels			
±0.1 V Range	48 channels	48 channels	48 channels		48 channels	51 channels			
±0.067 V Range	48 channels	48 channels	48 channels	48 channels	48 channels	51 channels	32 channels	44 channels	44 channels
±0.01 V Range	48 channels	32 channels [1,2,3]	16 channels [1,2,3]	48 channels	48 channels	51 channels	32 channels	44 channels	44 channels
TC Connectors	0	16	32	48	48	0	0	44	44
Internal CJCs [4]	0	4	8	12	12	0	8	11	11
External CJCs [5]	3	2	1	0	0	0	0	0	0
Voltage Channel	0	0	0	0	0	0	0	2 channels (0-300V AC)*	
Current Channel								2 channels (0-5A AC)*	

Notes

[1] D-sub connector inputs can be configured for all voltage ranges.

[2] D-sub connector inputs can be configured for thermocouple measurement provided the external CJC sensor is wired and activated.

[3] Thermocouple connector inputs can be configured for 0.067 V, 0.1 V, or thermocouple measurement. The EX1000A-TC is an exception, as it provides 48 thermocouple connectors capable of performing measurements on all voltage ranges indicated above.

[4] For thermocouple connector inputs, there is one internal CJC sensor for each group of four channels.

[5] For D-sub connector inputs, there is one external CJC sensor connection (10,000 Ω thermistor) for all 16 channels.

* For EX1044A accuracy refer SCM5B 33-05 & SCM5b33-07 Dataforth module website i.e. http://www.dataforth.com/signal-conditioning.aspx for voltage & current channels respectively

EX10XXA DIMENSIONAL DIAGRAM

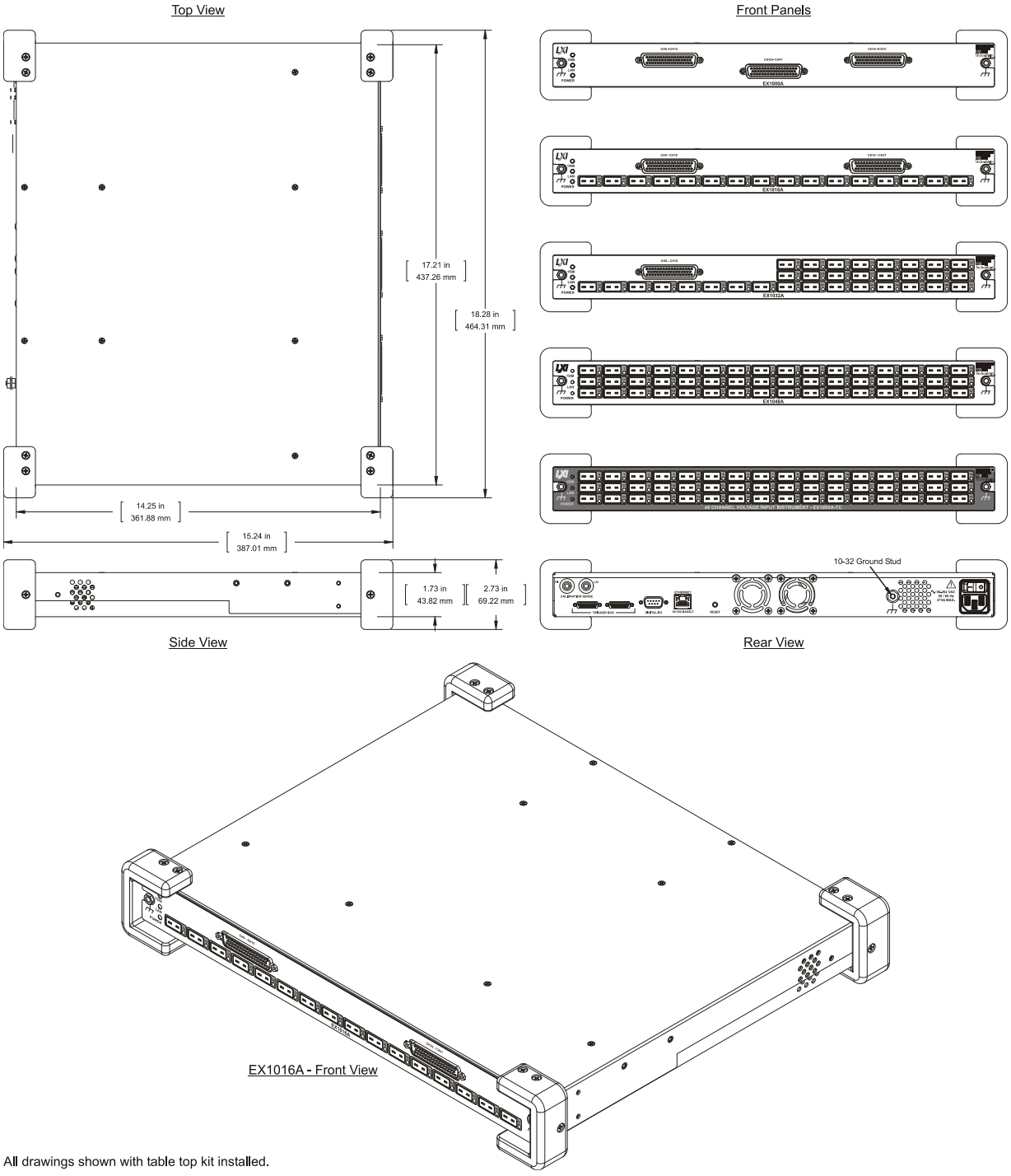


FIGURE 1-1: RACKMOUNT HARDWARE INSTALLATION DIAGRAM

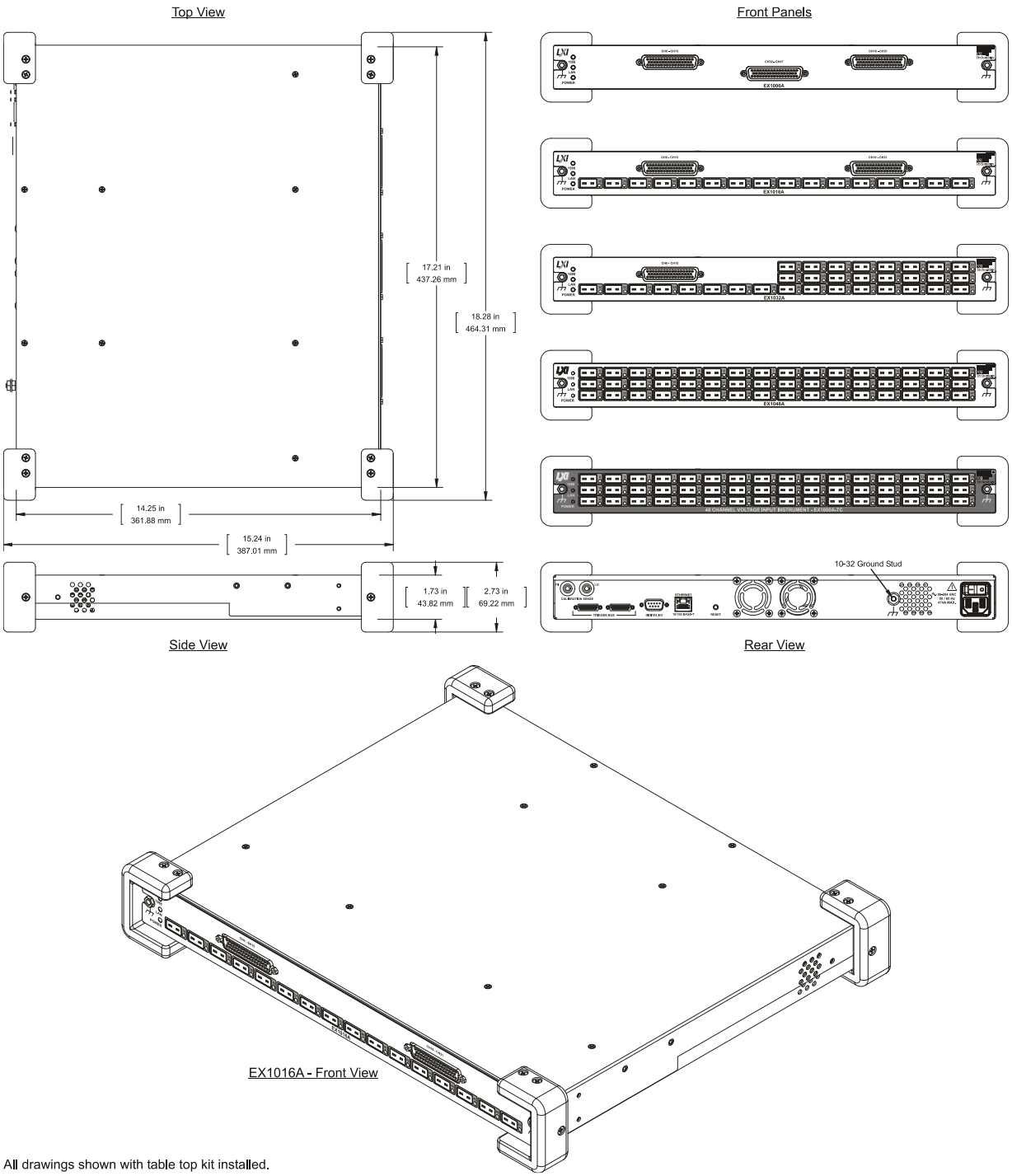
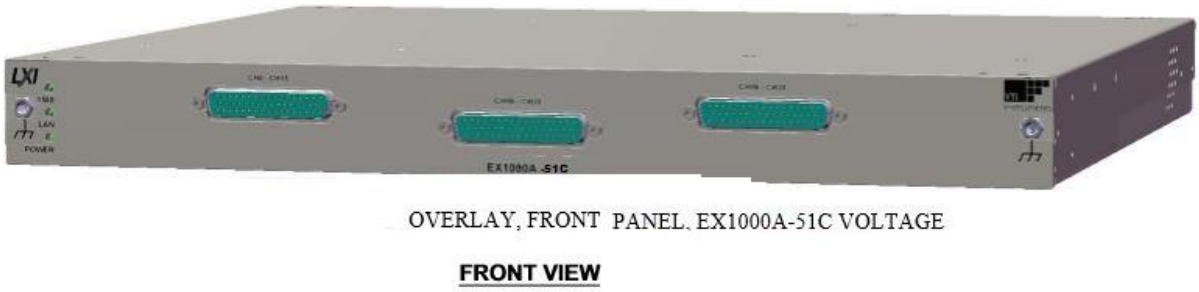


FIGURE 1-2: RACKMOUNT HARDWARE INSTALLATION DIAGRAM



OVERLAY, FRONT PANEL, EX1000A-51C VOLTAGE

FRONT VIEW

FIGURE 1-2A: EX1000A-51C

RX1032 DIAGRAM

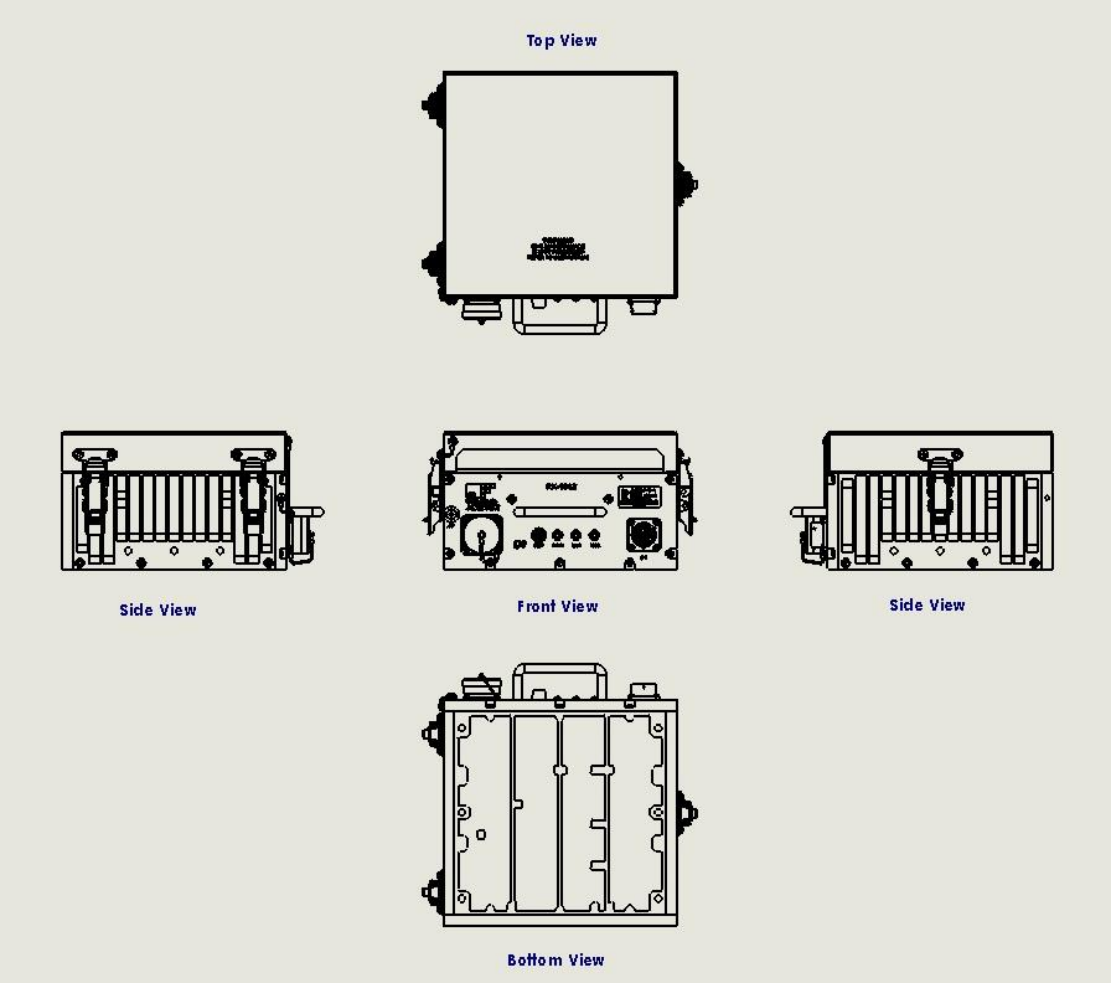
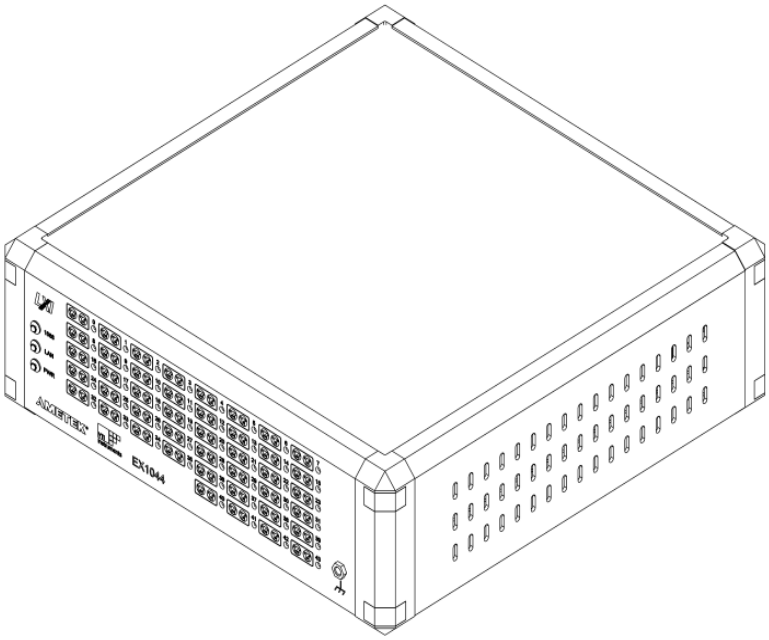


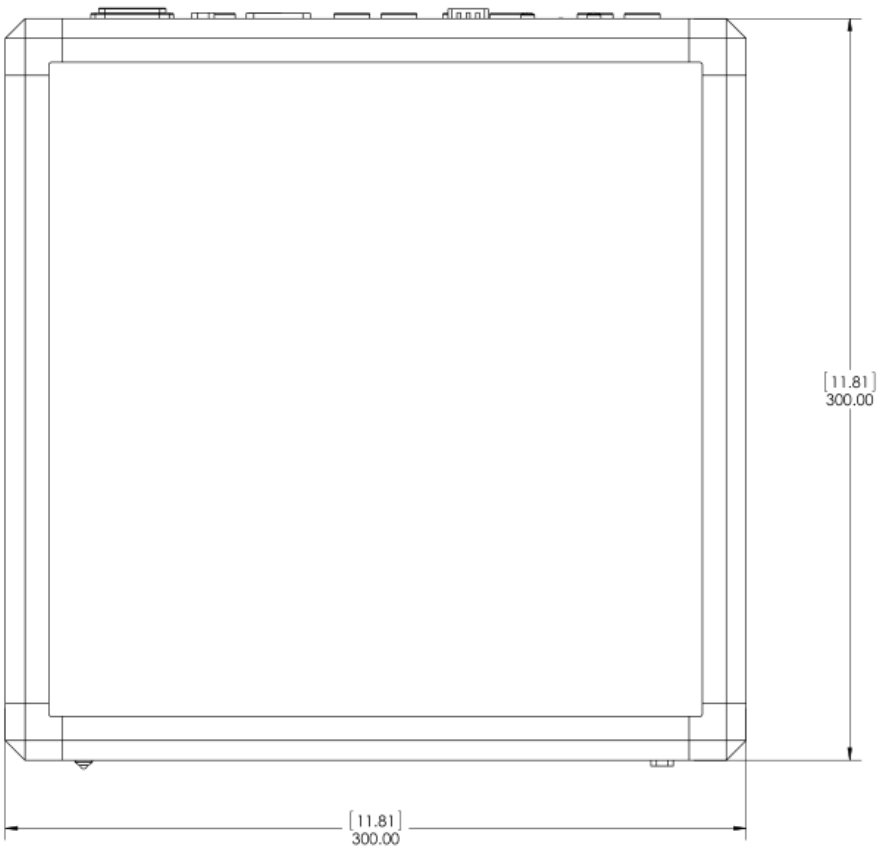
FIGURE 1-3: RACKMOUNT HARDWARE INSTALLATION DIAGRAM



FIGURE 1-4: ISOMETRIC AND INTERNAL CJC PLATE DIAGRAM

EX1044A/EX1044 DIAGRAM





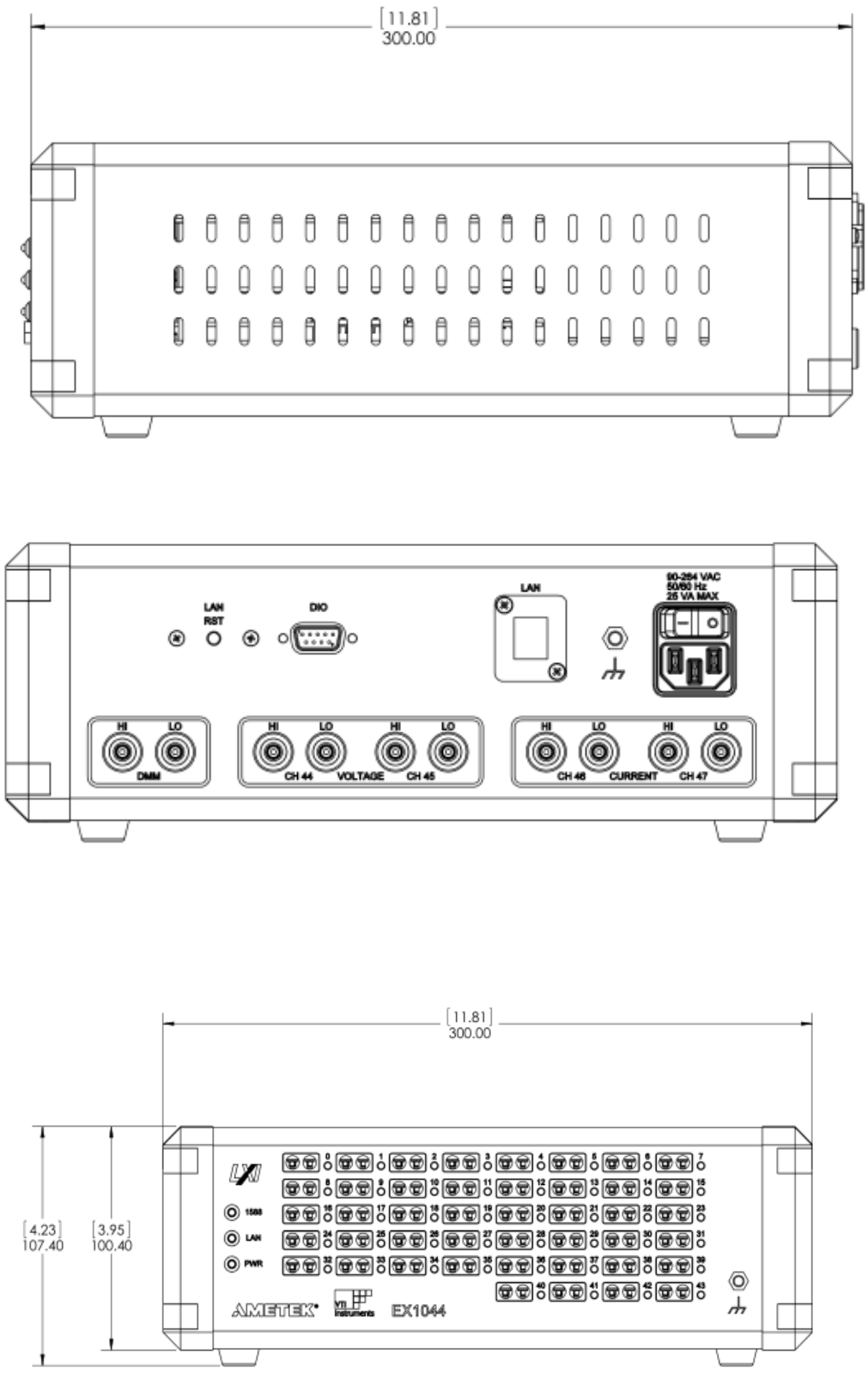


FIGURE 1-5: EX1044A/EX1044 TABLE TOP USAGE

EXPLANATION OF SPECIFICATIONS

The base accuracy of the EX10xxA is specified over an ambient operating temperature range of 20 ºC to 30 ºC and within one year of full calibration. This accuracy is shown in Table 1-2. Significant performance improvement over the base accuracy can be realized, however, using periodic self-calibration. Note, however, that the base ambient operating temperature range and full calibration time interval restrictions still apply.

The EX10xxA's thermocouple accuracy table provides the user with an easy-to-use description of the unit's capabilities at several points over the valid input dynamic range of the instrument. Each specific thermocouple type is represented by a different row. Each column refers to an input temperature of that value. For example,, an input temperature of 100 ºC on a type K thermocouple has a maximum instrument uncertainty of ±0.20 ºC, exclusive of thermocouple errors.

Exclusive of thermocouple errors

The "exclusive of thermocouple errors" qualification refers to two inherent sources of error present in all thermocouples to some extent. The first, and most obvious, error source is the accuracy of the thermocouple itself. This refers to the extent to which the specific thermocouple potentially deviates from the standardized International Temperature Scale of 1990 (ITS-90) characteristic for its thermocouple type. The second, less obvious error source relates to the resistance of the thermocouple wire, which creates a voltage drop against the input bias current of the EX10xxA. In most applications, this error effect is negligible. However, if the length and wire gauge of the employed thermocouple represents a resistance over 250 Ω, its effect should be analyzed for significance.

As an example, 650 ft of 24 gauge type T wire has a resistance of about 500 Ω. Against the EX10xxA's typical input bias current of 7.5 nA, this creates a voltage error of:

$$500\,\Omega \times 7.5\,\text{nA} = 3.75\,\mu\text{V}$$

To convert this error to its representative temperature error, it is then divided by the slope of the thermocouple characteristic at the temperature of interest. For example, the slope of the type T characteristic at 0 ºC is 39 µV/ºC. The error at this point is then:

$$3.75\,\mu\text{V} \div 39\,\mu\text{V}/^{\circ}\text{C} = 0.1^{\circ}\text{C}$$

Common mode rejection

The common mode rejection characteristics of the EX10xxA are specified in terms of dB for both dc and (50/60) Hz interference. It should be noted that these specifications refer to the distortion of the measurement in terms of voltage, not temperature. The following example illustrates how to determine the possible temperature error for a T type thermocouple measuring 0 ºC due to a 1 V dc common mode signal.

The EX10xxA has a minimum CMRR at dc of 100 dB. A 1 V common mode voltage creates a maximum differential voltage error of:

$$\frac{1\,\text{V}}{10^{\left(\frac{100}{20}\right)}} = 10\,\mu\text{V}$$

As before, this voltage error converts to a temperature error by dividing by the appropriate thermocouple slope to yield a maximum error of:

$$10\,\mu\text{V} \div 39\,\mu\text{V}/^{\circ}\text{C} = 0.26^{\circ}\text{C}$$

MAXIMIZING MEASUREMENT PERFORMANCE

This section discusses tips and procedures that can help maximize the actual performance realized with the EX10xxA and aid the user in avoiding some common pitfalls associated with thermocouple measurement.

Utilize self-calibration

Self-calibration should be conducted as often as practical, especially if the ambient environment has changed significantly since the previous calibration. However, fast ambient environmental changes should ideally be followed by a period of thermal stabilization before conducting self-calibration. The self-calibration process completes quickly and does not require removal of the actual input connections, making it convenient to run often.

NOTE	Self calibration is not applicable for RX10xx since during factory calibration process each instrument will be calibrated in precise manner and calibrated data gets stored in the instrument memory. The instrument has to be sent to factory to perform subsequent calibration after the initial calibration date is expired. *Refer below section for* ***Dynamic Calibration in RX1032***

Dynamic Calibration in RX1032

The dynamic calibration should be conducted to calibrate the measurement drift due to the temperature change. For every measurement scan, the values are adjusted based on the system internal temperature.

- The initial calculated Voltage and Temperature coefficients are stored into the RX1032 system.
- At every measurement the measured values are calibrated dynamically using the stored coefficients.
- Based on the present temperature of the system, the values are calibrated and reported.

Utilize self-test

Self-test should be conducted to check the working condition of CJC thermistor. Generally it should be followed after self-calibration However at least 30 minutes warm up time should be sufficient enough before conducting self-test. The self-test process takes at least 2 minutes for completion and does not require removal of the actual input connections, making it convenient to run often. After every self-test operation the unit should be stabilized for at least 30 minutes.

Allow for cold junction thermal stabilization

The performance of any thermocouple measurement instrument is largely determined by the stability of the cold junction sensing mechanism. For maximum accuracy and stability, the EX10xxA is designed with a significant thermal mass that connects the input connectors to the cold junction sensing element. However, while this lowers the sensitivity to thermal disturbances, transient measurement errors are still possible under certain operating conditions. Awareness of these conditions will help achieve the maximum performance from the EX10xxA.

Step change in ambient temperature

If the EX10xxA is subjected to a significant ambient temperature change over a short period of time, a transient measurement error will occur due to the inherent thermal time constant differences between the thermal mass and the actual cold junction made between the EX10xxA input connector and the type-specific thermocouple connector. The magnitude of the error will be directly proportional to the rate of change of the ambient temperature. Because the actual cold junction is more closely tied to the external ambient conditions than the internal thermal mass, it will follow the ambient temperature change to a slightly better extent than the thermal mass. Consequently, when subjected to a drastic rise in ambient temperature, the actual cold junction will rise in

temperature at a slightly faster rate than the thermal mass (and thus cold junction measurement). In this case, the channel measurements will show a transient error of negative polarity. Conversely, if the ambient temperature drastically falls, the channel measurements will show a transient error of positive polarity. In either case, as the rate of ambient temperature change decreases, thermal equilibrium is regained, and the error is eliminated. This error mechanism is not affected by calibration and is avoided by allowing for an appropriate thermal equilibrium delay.

NOTE	Significant ambient temperature changes should be followed by a thermal equilibrium delay before the initiation of measurements.

Initial insertion of a jack / screw terminal connection

A similar error mechanism is introduced when a male thermocouple jack / wire connected to screw terminal is first inserted into an EX10xxA/RX10xx input channel. A transient error is generated because the male thermocouple jack and mating female jack on the EX10xxA are at different temperatures when they make first contact. If both the unit and male jack are at the same ambient temperature, this error typically dissipates to less than 0.1 ºC within 3 minutes. However, a jack at room temperature plugged into a unit that is at a substantially different ambient temperature will generate a significantly larger initial error that takes longer to decay to a level of insignificance. This error mechanism is not affected by calibration and is avoided by allowing for a thermal equilibrium delay after initial jack insertion.

Insertion of jacks / screw terminal connection into neighboring channels

A less obvious source of thermal destabilization is the insertion of jacks into neighboring channels. Because each CJC sensor is shared among four input channels, there is a thermal disturbance seen by it when jacks are inserted into any of the four channels that it is monitoring. The disturbance occurs because the newly inserted jack represents a thermal mass that is at a different temperature than the internal thermal mass. This instantly lowers the measured CJC temperature to a small extent. Since the same CJC temperature is used for three other channels, they exhibit a small transient error of negative polarity as a result. The magnitude of the error is proportional to the quantity of jacks that are inserted at a time, with three additional jacks being the worst-case scenario. The error decreases with time, as thermal equilibrium is gradually restored. Empirical testing has shown that the worst-case error typically dissipates to less than 0.1 ºC within 3 minutes. This error mechanism is only present on channels that share a CJC sensor. Moreover, it is essentially not present when jacks are removed, as no thermal disturbance is created. This error mechanism is not affected by calibration and is avoided by allowing for a thermal equilibrium delay after jack population changes. For reference, the association between CJC channels and input channels is shown in Table 1-3.

NOTE	Jack population changes should be followed by a thermal equilibrium delay before the initiation of measurements.

CJC #	Channel	Input Channel
CJC0	48	0-3
CJC1	49	4-7
CJC2	50	8-11
CJC3	51	12-15
CJC4	52	16-19
CJC5	53	20-23
CJC6	54	24-27
CJC7	55	28-31
CJC8	56	32-35
CJC9	57	36-39

CJC10	58	40-43
CJC11	59	44-47
CJC12	60	48-51

TABLE 1-3: CJC CHANNEL / INPUT CHANNEL RELATIONSHIP

Select the proper hardware filter

Unless the bandwidth of the sensor requires a higher instrument bandwidth, the 4 Hz setting of the EX10xxA hardware filter should be used, as it provides the greatest immunity to external electrical and magnetic interference.

Choose an appropriate sampling rate

For best instrument noise performance, the sampling rate should be set as low as the data collection requirements allow. For more details, see *Sampling Rate* / Noise Performance in Section 3.

Select the proper location

The EX10xxA unit should be located away from sources of high or low temperature, strong air currents, and high magnetic fields. For more details, see Installation in Section 2

Use the correct wiring

Best results will be achieved with the shortest and largest thermocouple wire that the physical requirements of the application can support. In addition, shielded thermocouples can be employed to raise the system's rejection of electrical interference. For more details, see *Input Connections / Wiring* in Section 2.

SECTION 2

PREPARATION FOR USE

UNPACKING EX10XXA, EX1000A-51C

When the EX10xxA is unpacked from its shipping carton, the contents should include the following items:

- EX10xxA Precision Thermocouple Instrument
- LXI Quick Start Guide
- EX10xxA User's Manual (this manual)
- VTI Instruments Corp. Drivers and Product Manuals CD
- Power line cord

All components should be immediately inspected for damage upon receipt of the unit.

UNPACKING RX10XX

When the RX10xx is unpacked from its shipping carton, the contents should include the following items:

- RX10xxA Precision Rugged Thermocouple Instrument
- CD containing all the required software & documents
- Mating connector for P1 13-Pin circular connector and J1 RJ45 mating shell connector
- Calibration certificate
- RX1032 Quick Guide Hardcopy

CD Contents:

- LXI Quick Start Guide
- EX10xxA/RX10xx User's Manual (this manual)
- Product Drivers and Product Manuals CD
- EXLab software/ Manual (Optional)

It also contains IVI shared components required to be installed for the device identification purpose. LXI discovery utility & Bonjour setup are provided in the CD which is helpful to discover the RX1032 and other LXI device connected in the network.

All components should be immediately inspected for damage upon receipt of the unit.

UNPACKING EX1044A/EX1044

When the EX1044A/EX1044 is unpacked from its shipping carton, the contents should include the following items:

- EX1044A/EX1044 Precision Rugged Thermocouple, Voltage/Current Instrument
- CD containing all the required software & documents
- Mating connector for P1 13-Pin circular connector and J1 RJ45 mating shell connector
- Calibration certificate

- EX1044 Quick Guide Hardcopy

CD Contents:

- LXI Quick Start Guide
- EX10xxA/RX10xx/EX1044 User's Manual (this manual)
- Product Drivers and Product Manuals CD
- EXLab software/ Manual (Optional)

It also contains IVI shared components required to be installed for the device identification purpose. LXI discovery utility & Bonjour setup are provided in the CD which is helpful to discover the EX1044 and other LXI device connected in the network.

All components should be immediately inspected for damage upon receipt of the unit.

NOTE	IO Library Suite (formerly called as Agilent Connection Expert) (optional: used for instrument discovery if Bonjour is not used. The latest version of IO Libraries can be downloaded from the following Keysight website address i.e. http://www.keysight.com/find/iosuiteproductcounter

INSTALLATION LOCATION

The EX10xxA is designed to be largely insensitive to external electrical, magnetic, and thermal disturbances. However, as with all precision instrumentation, certain precautions, if taken into consideration, can help achieve maximum performance.

1) The unit, particularly its front panel, should be located away from sources of high or low temperatures. When used in a rack-mount application with other heat-generating instruments, the EX10xxA should be located as far away from the other instruments as possible, 1U minimum. Multiple EX10xxAs, however, can be stacked directly on top of one another without any performance degradation.
2) The front panel of the EX10xxA should not be exposed to strong air currents. Typical problematic sources include building ventilation and instrument or cabinet fans.
3) The unit should be located away from sources of high magnetic fields such as motors, generators, and power transformers.

INSTALLATION OPTIONS

The EX10xxA has three options which can installed prior to installation:

- Rackmount Installation Option (P/N: 70-0355-900)
- Rack Ear Installation Option (P/N: 70-0355-901)
- Tabletop Installation Option (P/N: 70-0355-902)

These options are not included with the EX10xxA and must be ordered separately.

Rackmount Installation Option of EX10xxA

The rackmount installation option includes all the parts necessary to mount the EX10xxA to the front and rear of a standard test rack.

Required Tools

1) #2 Phillips screwdriver
2) 5/16" wrench or socket

Parts List

Item#	Qty	Description	VTI P/N
1	2	Rack Ear, EX10XXA	41-0482-009

2	2	Bracket, Unit Support, EX10XXA	41-0482-010
3	2	Bracket, Rear Rack Support, EX10XXA	41-0482-011
4	10	Screw, 6-32 x 3/8", Flat Head Phillips, 82°, Stainless Steel	37-0173-037
5	2	Screw, 6-32 x 3/8" Pan Head Phillips, Sq Cone SEMS Zinc	37-0028-037
6	6	Screw, 8-32 x 3/8" Pan Head Phillips, Sq Cone SEMS Zinc	37-0073-037
7	4	Nut, Hex, KEPS, 8-32 Steel/Zinc	37-0200-832

Assembly Procedure

1) Place the chassis on a protected work surface with its input connectors facing front.
2) Using a #2 Phillips screwdriver, install the rack ears (P/N: 41-0482-009) on the front of the EX10xxA using five (5) 6-32 X 3/8" flat head Phillips stainless steel screws (P/N: 37-0173-037) for each rack ear. Note that the "Optional Mounting Locations" in Figure 2-1 allow the EX10xxA to be installed in a rack either flush or recessed.
3) Next, attach the unit support brackets (P/N: 41-0482-010) to the EX10xxA using two (2) 6-32 x 3/8" pan head Phillips SEMS screws (P/N: 37-0028-037) for each bracket (see Figure 2-1).
4) Place the rear rack support brackets (P/N: 41-0482-011) on the inside of the unit support brackets installed in step 3 and loosely secure them using two (2) 8-32 x 3/8" pan head Phillips SEMS screws (P/N: 37-0073-037) and two (2) KEPS hex nuts (P/N: 37-0200-832). Loosely installing the hardware will allow the rear support bracket to adjust to the proper length once installed in a rack.
5) Place the EX10xxA in the desired rack slot and secure the front rack ears with hardware provided with the rack. Once the front bracket are secured, slide the rear support brackets so that they can be attached to the rack as well. After securing the rear support brackets, tighten the screws and KEPS nuts installed in step 4.

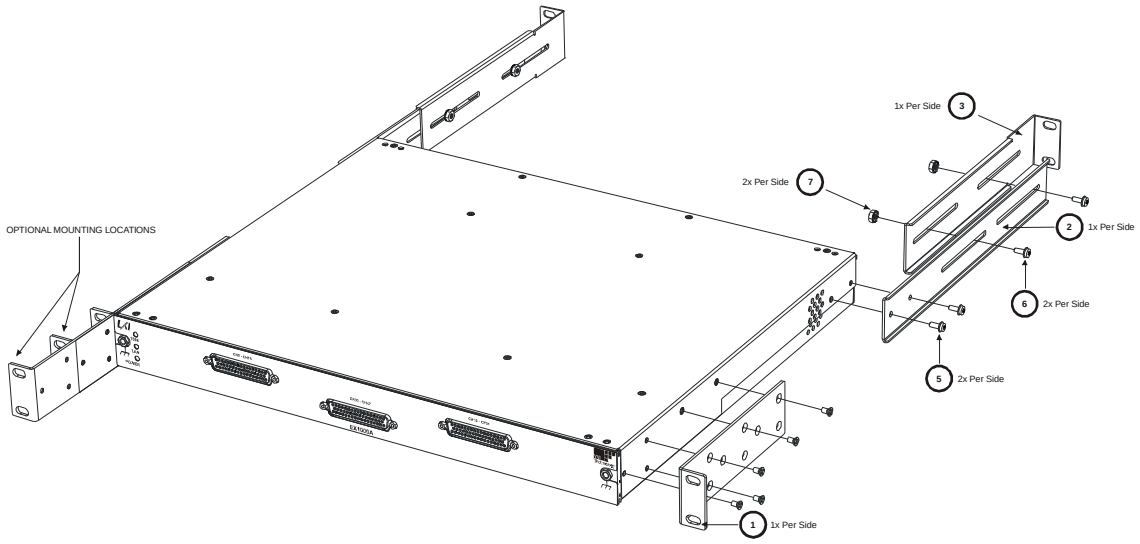


FIGURE 2-1: RACKMOUNT HARDWARE INSTALLATION DIAGRAM

Rack Ear Installation Option

The rackmount installation option includes all the parts necessary to mount the EX10xxA to the front of a standard test rack.

Required Tools

1) #2 Phillips screwdriver

Parts List

Item#	Qty	Description	VTI P/N
1	2	Rack Ear, EX10XXA	41-0482-009
2	10	Screw, 6-32 x 3/8", Flat Head Phillips, 82°, Stainless Steel	37-0173-037

Assembly Procedure

1) Place the chassis on a protected work surface with its input connectors facing front.
2) Using a #2 Phillips screwdriver, install the rack ears (P/N: 41-0482-009) on the front of the EX10xxA using five (5) 6-32 x 3/8" flat head Phillips stainless steel screws (P/N: 37-0173-037) for each rack ear. Note that the "Optional Mounting Locations" in Figure 2-2 allow the EX10xxA to be installed in a rack either flush or recessed.

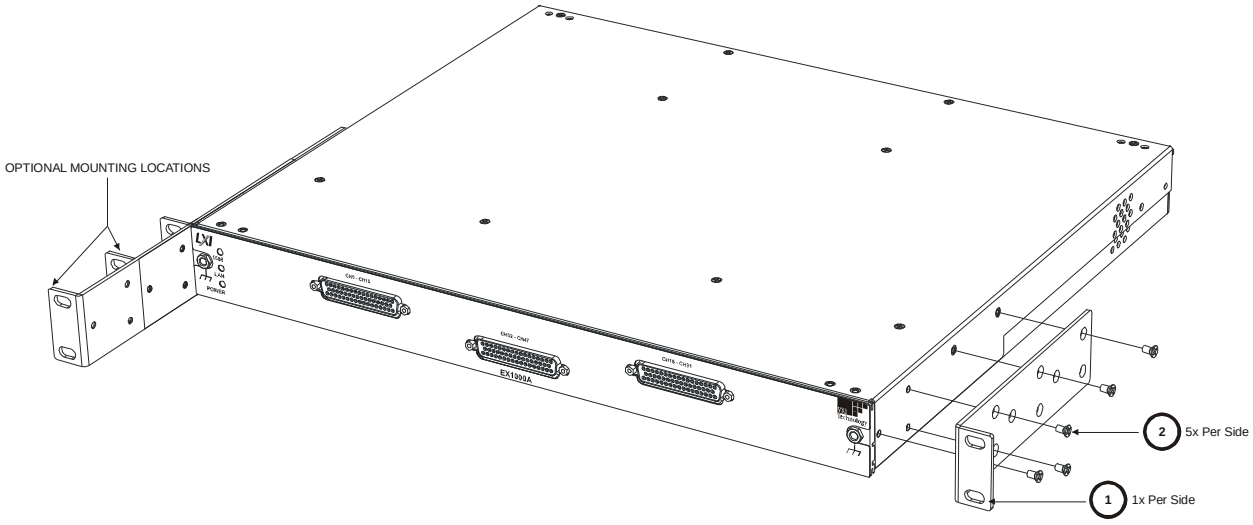


FIGURE 2-2: RACK EAR INSTALLATION DIAGRAM

Tabletop Installation Option

The tabletop installation option can be used when the EX10xxA is will not be installed in a rack, but will be employed as a bench top or desktop instrument.

Required Tools

1) #2 Phillips screwdriver

Parts List

Item#	Qty	Description	VTI P/N
1	16	Screw, 6-32 x 7/16", Pan Head Phillips, Zinc	37-0044-044
2	4	Screw, 6-32 x 1/2", Pan Head Phillips, Zinc	37-0044-050
3	4	Feet, Corner, Rubber, "U" Shaped	37-0281-201

Assembly Procedure

1) Place the chassis on a protected work surface with its input connectors facing front.
2) Using Figure 2-3, locate the installation locations for each rubber foot.
3) Using the #2 Phillips screw driver, install the four rubber feet using four (4) 6-32 x 7/16" pan head Phillips screws to secure the top and bottom and one (1) 6-32 x 1/2" pan head Phillips screw to secure side of each foot.

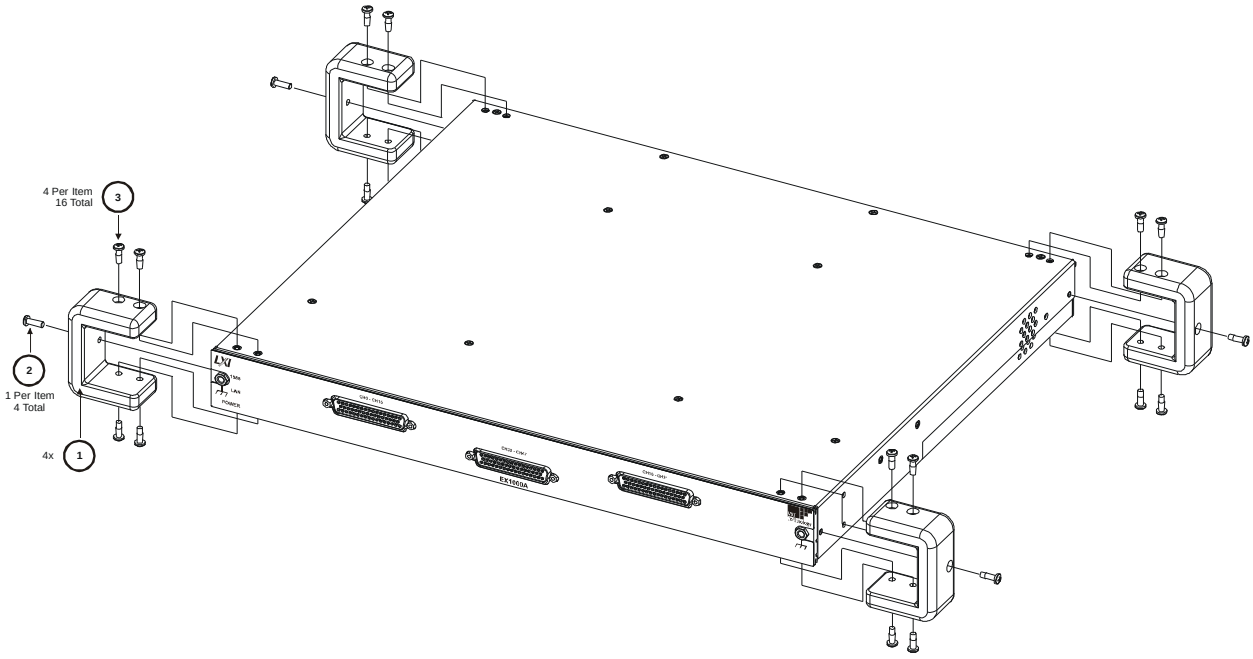


FIGURE 2-3: TABLE TOP FEET INSTALLATION DIAGRAM

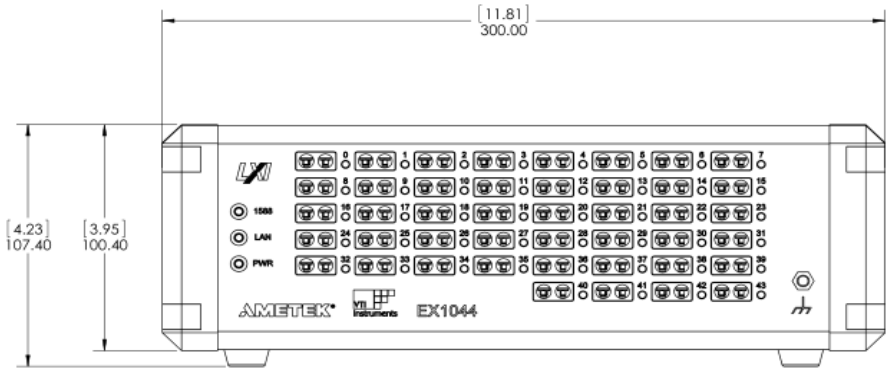


Figure 2-4: EX1044A/EX1044 Table Top USAGE

Installation Option of RX10xx

Mechanical Accessories

1) Power Connector: 27-0673-315, CONN, MIL-C-26482-1, CABLE PLUG, SIZE 14, 15 WAY, SOCKET
2) Ethernet Connector: 27-0926-008I, CONN, IP67, RJ-45, PL, 8POS, ST
3) Mounting Angle Bracket: 41-0604-012 (2 ea)
4) Screw, M6X10mm: 37-1901-010 (6 ea)
5) Cable Mount Bracket: 41-0604-013 (2 ea)
6) Screw, M3X6mm: 37-0223-006 (4 ea)

NOTE: Mechanical accessories need to order separately

Assembly Procedure

7) Mount option-I: 6X M6 tapped holes are provided at the bottom of the chassis as per Figure 2-3-1 for fixing on horizontal plate or with vibration mount.
8) Mount option-II: The System using Left and Right angle plates provided in the chassis.
9) Mount the unit to the protected area with mounting details provided as per Figure 2-3-1
10) Hole size in the mounting angle bracket is 6.5 mm (dia). Use 6 X M6 X 10mm LG, CAP FLANGE HD, 10.9 STEEL SCREWS

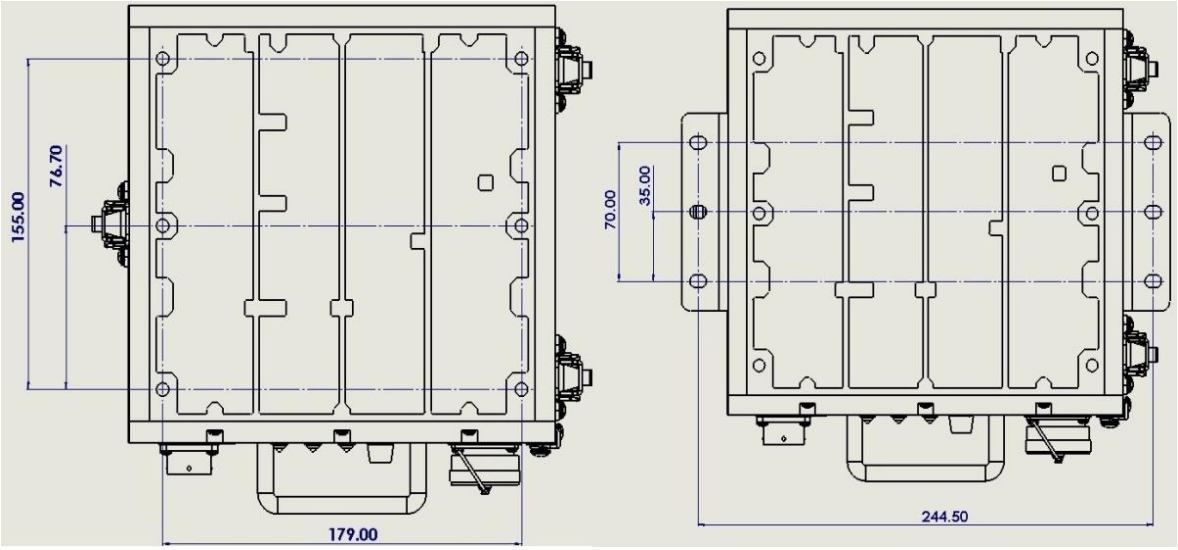


FIGURE 2-3-1: RX10XX CHASSIS MOUNTING DETAILS

11) Remove the Top Cover by opening 3 Latches on right and left side of chassis.



FIGURE 2-3-2: RX10XX CHANNEL SCREW TERMINAL PLATE VIEW

12) Unscrew the captive screws to inserts the thermocouple wires/ U - Lugs into it.
13) Fix the thermocouple 'U' Lugs into screw terminals. Each channel has S,+ & - signals screws (‘S’ for Shield, ‘+’ for Positive input, ‘-‘ for Negative input). Refer the below channel layout diagram of screw terminals.

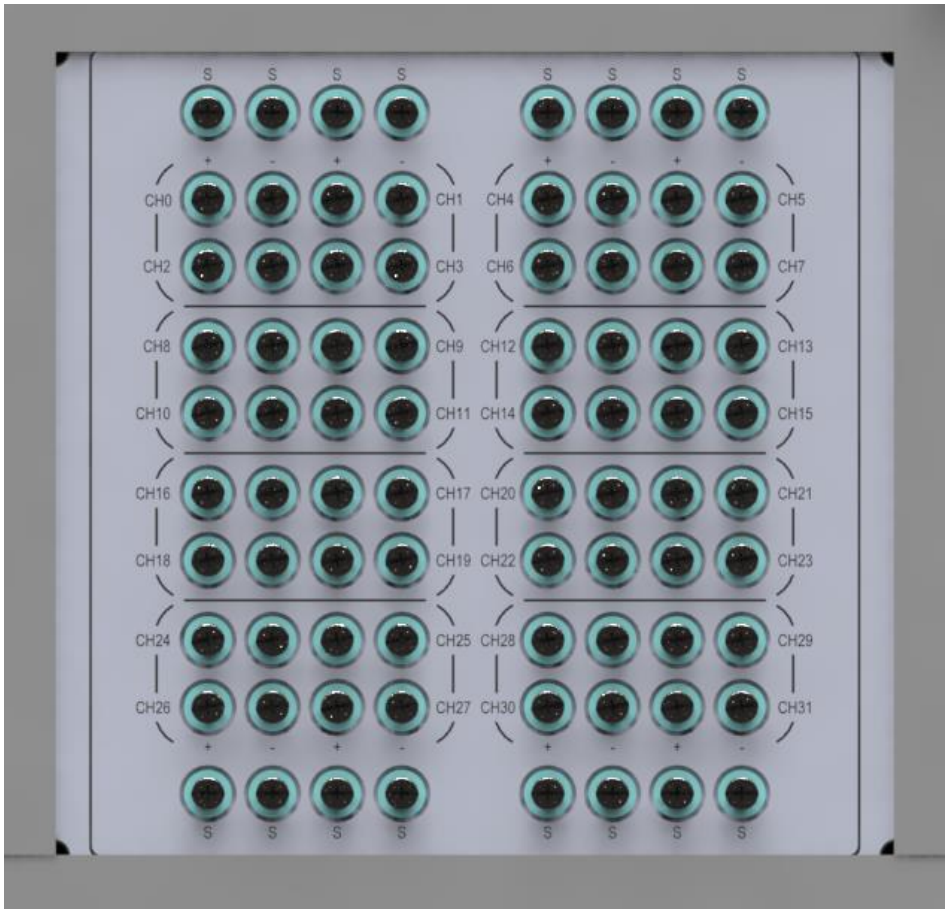


FIGURE 2-3-3: RX10XX CJC PLATE TOP VIEW DETAILS

14) Connect RJ45 Ethernet cable connector to J1 connector and other end to PC
15) Prepare P1 mating connector cable loom as per the "RX10xx Connections" Section
16) Connect P1 mating connector cable loom and other end +28V_IN and +28V_RET signals into 28V DC power supply. Power Supply should be capable of produce 2Amps min.
17) Power ON the supply.
18) Wait for 2min time to system boot.
19) Run the SFP application software and configure the channel as per requirement.
20) Wait for warm up time as mentioned in "WARMUP TIME" section
21) Start acquire the channel data

WARM-UP TIME

The specified warm-up time of the EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044 is 60 minutes. If, however, the unit is being subjected to an ambient temperature change greater than 5 ºC, extra stabilization time is recommended to achieve maximum performance.

SOFTWARE INSTALLATION

The fastest way to begin controlling an EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044 is to discover the unit using LXI Discovery utility. To do this, the following is required:

- A PC or laptop computer
- A Microsoft™ Windows™ XP OS with Service Pack 2 (SP2) and Internet Explorer™ (IE)

- An Internet connection
- LXI Discovery utility (Available at LXI consortium website http://lxistandard.org/Resources/LXIDiscoveryTool.aspx)

The LXI Discovery tool is a free utility, available for download from LXI Consortium itself. It uses both VXI-11, as well as mDNS protocols to detect the LXI instruments present in your network. Once installed, this software will discover EX10xxA instruments, if available, as shown below.

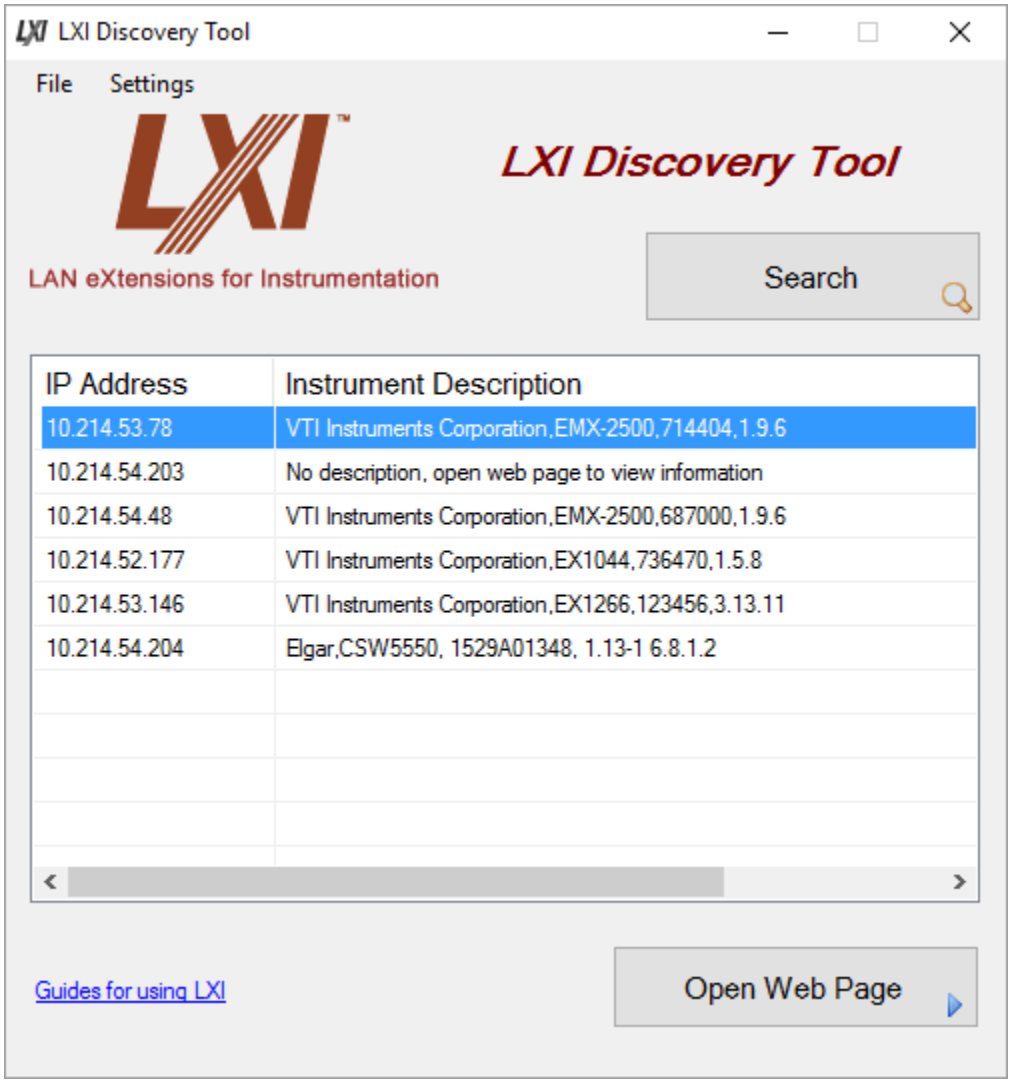


FIGURE 2-4: LXI DISCOVERY TOOL LIST ALL THE INSTRUMENTS

Alternatively, the EX10xxA may also be discovered using Internet Explorer's Bonjour for Windows plug-in. The IP address of the EX10xxA instrument can also be entered into the address bar of any web browser to view the embedded web page. By default, the EX10xxA will first attempt to use DHCP to set its IP Address. If DHCP is not available on the network it is connected to, it will instead use Auto IP. Determining the Auto IP address is discussed in below sections. Other discovery methods, such as VXI-11, can be used as well.

For more information on discovering the EX10xxA, please refer to *Web Page Operation* in *Section 4*.

NETWORK CONFIGURATION

By default, the EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044 will attempt to locate a DHCP server. If one is found, the IP address assigned by the DHCP server will be assumed. Otherwise, after a timeout of 20 seconds, the unit will attempt to obtain an IP address by using Auto IP.

Auto IP is a mechanism for finding an unused IP address in the range 169.254.X.Y where X is in the range 1 - 254 and Y is in the range 0 - 255. The device will first attempt to obtain the specific address 169.254.X.Y, where X and Y are the second-to-last and last octets of the device's MAC address. However, X will be set to 1 if it is 0 in the MAC address, and to 254 if it is 255 in the MAC address. If this address is already in use, the unit will attempt to obtain other IP addresses in a pseudorandom fashion until it finds one that is available.

To illustrate the Auto IP mechanism, Table 2-1 lists the Auto IP default address for some example MAC addresses.

MAC Address	Auto IP Default Address
00:0D:3F:01:00:01	169.254.1.1
00:0D:3F:01:01:01	169.254.1.1
00:0D:3F:01:A3:28	169.254.163.40
00:0D:3F:01:FE:FE	169.254.254.254
00:0D:3F:01:FF:FE	169.254.254.254

TABLE 2-1: AUTO IP DEFAULT ADDRESS ASSIGNMENT

If a static IP address assignment is preferred, one can be optionally assigned via the embedded web page interface. This is done by clicking the **Network Configuration** link, disabling DHCP, and then assigning a static IP address. For more information, see *Network Configuration* in Section 4.

However, a much more convenient and recommended way to obtain the benefits of a static IP address is to employ DHCP, but assign the instrument a reserved IP address in your company's DHCP server configuration. This reserved address, linked to the EX10xxA's MAC address on the DHCP server, would be assigned to the EX10xxA at power up initialization without having to manually set it on the EX10xxA. The DHCP server configuration provides a centralized, controlled database of assigned IP addresses, preventing accidental assignment of the same IP address to multiple instruments. Consult your company's Information Technology department for assistance.

VXI-11 Device Discovery is also supported by the EX10xxA. This allows all EX1048s on a local network to be found without knowledge of their MAC address or IP address with the use of a broadcast message.

Reset button for EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044

The reset button on the rear panel of the EX10xxA can be used to restore default network settings. This is useful for recovery from an incorrect or unknown network configuration. To perform a network reset:

1) Power off the EX10xxA.
2) Press and hold the reset button.
3) Power on the EX10xxA.
4) Continue to hold the reset button for at least 30 seconds.
5) Release the reset button.

The EX10xxA will power up as usual, but will use the default network configuration (DHCP) instead of its previous settings.

NETWORK TROUBLESHOOTING

If an error occurs when trying to discover the EX10xxA (see *Web Page Operation* in *Section 4* for more information on discovery), it may be necessary to change the network settings for the EX10xxA and the host PC. By using the following methodology, most network-related issues can be resolved:

1) *Restore the EX10xxA's Default Network Settings*
2) *Determine PCs Network Settings*
3) *Set the EX10xxA to Auto IP* or *Set the EX10xxA to Static IP*
4) *Restore the Host PCs Network Settings*
5) *Using Multiple Network* Cards

Restore the EX10xxA's Default Network Settings

It may be the case that the EX10xxA is in an unknown network configuration. The EX10xxA can be returned to its default state by pressing the *Reset button* located at the rear of the chassis.

1) Power off the EX10xxA.
2) Press and hold the reset button.
3) Power on the EX10xxA.
4) Continue to hold the reset button for at least 30 seconds.
5) Release the reset button.

Determine PCs Network Settings

1) Ensure that all host PC Ethernet connections are made that will be used while controlling the EX10xxA. If the PC will be connected to a LAN, ensure that the LAN connection is made.
2) From Windows , navigate to **Start→Settings→Network Connections**.
3) Right click on the connection that will be used to communicate with the EX10xxA, then select **Properties** (see Figure 2-5). Some PCs have multiple network connections, so it is important to make certain that the connection used to communicate with the EX10xxA is selected.

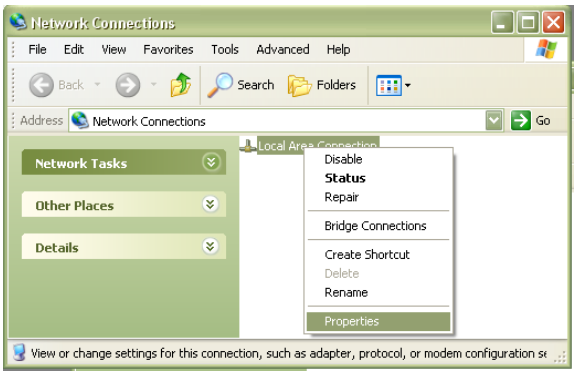


FIGURE 2-5: NETWORK CONNECTION PROPERTIES

4) Select **Internet Protocol (TCP/IP)**, then click the **Properties** button as shown in Figure 2-6.

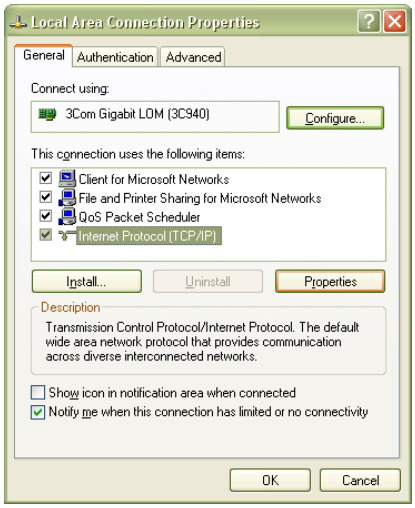


FIGURE 2-6: LOCAL AREA CONNECTION PROPERTIES DIALOG BOX

5) Determine if the PC is set to use auto or static IP. Figure 2-7 shows examples of both auto and static IP address configurations.

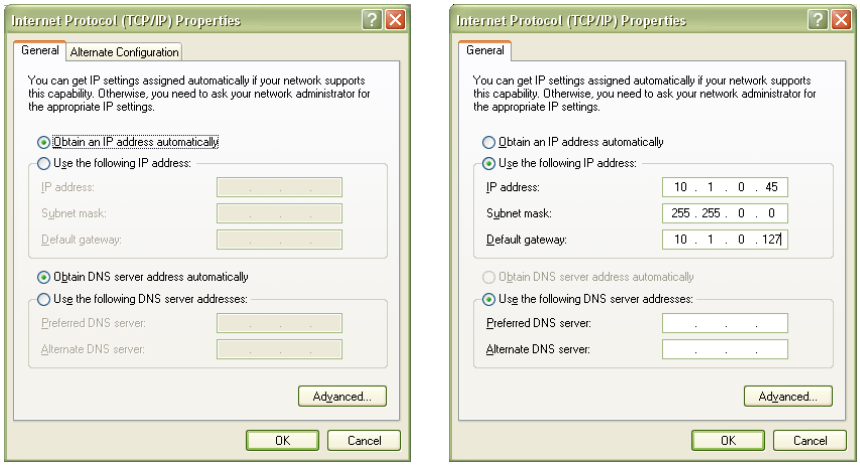


FIGURE 2-7: DYNAMIC (LEFT) AND STATIC (RIGHT) IP ADDRESS CONFIGURATIONS

6) If set to use a static IP, record the IP address, subnet mask, and default gateway for use later. Select **Obtain an IP address automatically** to establish a connection to the EX10xxA. Click the **OK** button and proceed to *Set the EX10xxA to Static IP*. If set to use a dynamic IP, click the **OK** button and proceed to *Set the EX10xxA to Auto IP*.

Set the EX10xxA to Auto IP

1) Remove all network connections from the PC except for the connection to the EX10xxA. Wireless adapters should be disabled as well.
2) Apply power to the EX10xxA and wait for the LAN LED to turn green.
3) Discover EX10xxA using Bonjour as described in the *Web Page Operation* in *Section 4*. The steps taken previously should ensure that discovery works.
4) Once connected to the EX10xxA, click on **Network Configuration** in the command menu.
5) Select **DHCP** and **AutoIP** in the **IP Address Source** field and ensure that **Static IP** is not selected. Figure 2-8 shows the proper configuration. Click the **Submit** button to save all changes.

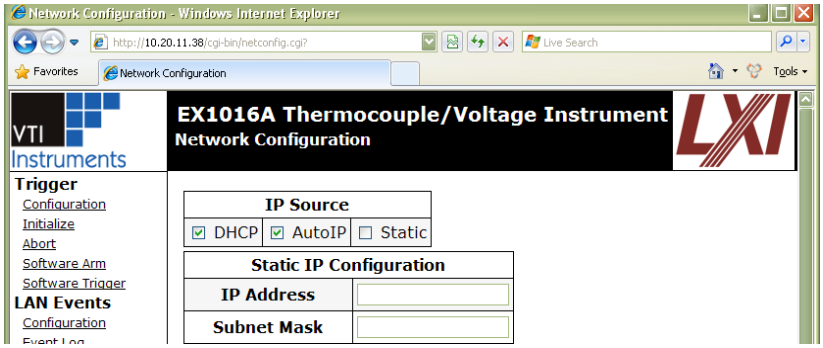


FIGURE 2-8: EX10XXA SET TO USE AUTO IP

Set the EX10xxA to Static IP

1) Disconnect all network connections from the PC except for the connection between the PC and the EX10xxA.
2) Apply power to the EX10xxA and wait for the LAN LED to turn green.
3) Discover EX10xxA using Bonjour as described in the *Web Page Operation* in *Section 4*. The steps taken previously should ensure that discovery works.
4) Once connected to the embedded web interface, click on the **Network Configuration** link in the command menu.
5) Select the **Static** checkbox in the **IP Address Source** field and ensure that **DHCP** and **AutoIP** are not selected.
6) In the **IP Address** field, enter an appropriate IP address for the EX10xxA. Use the IP address obtained the *Determine PCs Network Settings.*
7) step to determine the network address. The network address is the first three digits of the IP address (10.1.0 in the example provided). The last digit of the IP address (the node), is a number, 0 through 255, that is not currently assigned to any other device on the network. Ensure that a unique IP address is assigned to the EX10xxA by consulting a network administrator.
8) In the **Subnet Mask** field, enter the subnet mask in the *Determine PCs Network Settings*
9) step.
10) In the **Gateway Address** field, enter the same gateway address in the *Determine PCs Network Settings* step.
11) Click **Submit** to save all changes made to the **Network Configuration** page.

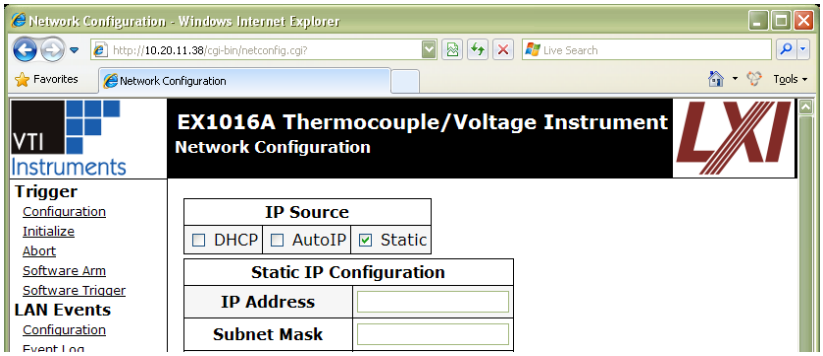


FIGURE 2-9: EX10XXA SET TO USE STATIC IP

Restore the Host PCs Network Settings

1) Power down the EX10xxA and reconnect it to the LAN in its desired location.
2) If the PC was originally set to use a static IP address, use the procedure in *Determine PCs Network Settings* to change the PCs IP address back to its original state.

3) Power on the EX10xxA and wait for the LAN LED to turn solid green.
4) Discover the EX10xxA using LXI Discovery as described in *Web Page Operation* in *Section 4*.

Using Multiple Network Cards

When multiple network cards exist in a single PC, it may be necessary to define a static IP address to both the host PC NIC card that will interface with the EX10xxA mainframe as well as the EX10xxA itself. This process is only necessary if a DCHP server is not connected to the network to which the device is connected and typically occurs when the NIC is connected directly to the instrument.

The following process can be used to ensure proper functionality.

1) Navigate to **Start → Settings → Network Connections**.
   - Disable all network interfaces except the one that is connected to the EX10xxA mainframe. This is done by right clicking on the interface, then selecting **Disable**.
   - Open the web page of the EX10xxA mainframe.
   - Click the **IP Configuration** link. A prompt may appear to log into the EX10xxA mainframe.
   - Unselect **DHCP** and **AutoIP** and ensure that **Static** is selected.
   - Enter an IP address into the **IP Address** field. Although any valid network IP address can be used, **192.168.1.2** is used in this example. For more information on valid IP addresses, please consult with an IT administrator.
   - Set the **Subnet Mask**. For this example, the subnet mask is **255.255.255.0**.

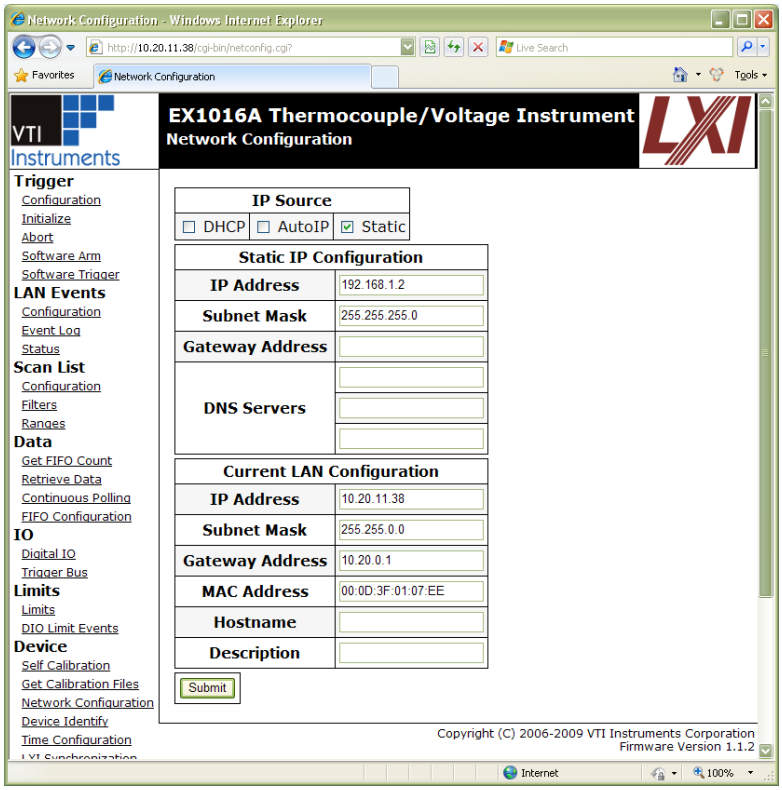


FIGURE 2-10: COMPLETED EX10XXA MAINFRAME STATIC IP CONFIGURATION

   - Click the **Submit** button. Once this is done, it is no longer possible to communication with the EX10xxA mainframe. This is normal and is addressed in the following steps.
   - Set a static IP address for the NIC card by doing the following:

a) Navigate to Start → Settings → Network Connections.
b) Right click on the NIC card that the EX10xxA mainframe is connected to and select **Properties**.
c) Select Internet Protocol (TCP/IP) and click **Properties**.

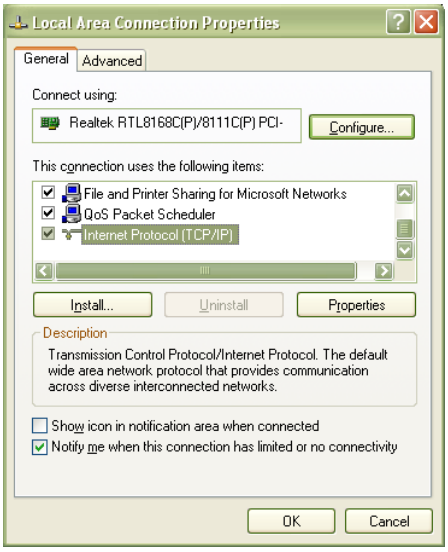


FIGURE 2-11: TCI/IP SELECTION

d) Click the Use the following IP address radio button.
e) Enter the desired IP address. If using the IP from the example above, **192.168.1.1** can be used.
f) If not automatically completed after the IP address is entered, set the **Subnet mask** field to **255.255.255.0**.

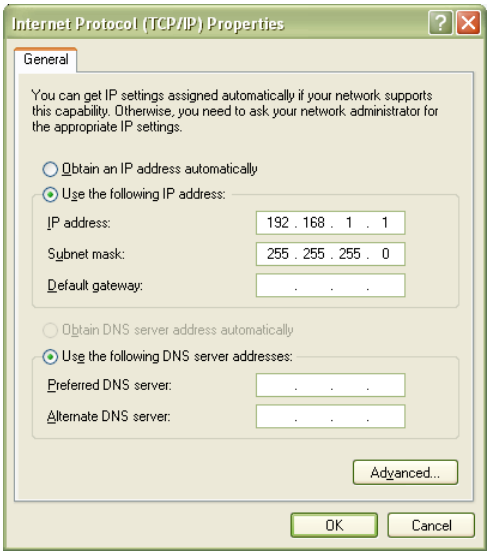


FIGURE 2-12: COMPLETED NIC STATIC IP CONFIGURATION

g) Click OK to exit the network configuration properties.

TIME CONFIGURATION

The EX10xxA will initially be configured to receive its time through PTP (Precision Time Protocol). The user can also set time using SNTP (Simple Network Time Protocol) or the time can be set manually. The manual setting is necessary if the network environment is such that the unit cannot reach the Internet. For more information, see *Time Configuration* in Section 4.

INPUT CONNECTIONS / WIRING

Thermocouple Connections

The EX10xxA employs an uncompensated (Cu-Cu) mini-thermocouple female jack as its input connector. This connector provides a solid, reliable connection that is also easily changeable. Since it is not thermocouple-type specific, different thermocouple types can be mixed throughout the unit without hardware modification. The input jack is polarized and will only accept its mating connector in one orientation. The mating connector is a standard mini-thermocouple male jack. A popular source is the SMPW series from Omega Engineering. For specified accuracy performance, the input connector must be of the same thermocouple type as the wire being connected.

Thermocouple wire is polarized, and it is critical to consider this polarity when connecting the thermocouple wire to the thermocouple jack. For reference, the color designations and polarizations of the most popular thermocouple types are listed in Table 2-2 for both ANSI (American) and IEC (European) standards.

ANSI Thermocouple Standard				
Thermocouple	**+**		**-**	
Type J	White		Red	
Type K	Yellow		Red	
Type T	Blue		Red	
Type E	Violet		Red	
Type S	Black		Red	
Type R	Black		Red	
Type B	Gray		Red	
Type N	Orange		Red	

IEC Thermocouple Standard				
Thermocouple	**+**		**-**	
Type J	Black		White	
Type K	Green		White	
Type T	Brown		White	
Type E	Violet		White	
Type S	Orange		White	
Type R	Orange		White	
Type B	Gray		White	
Type N	Pink		White	

TABLE 2-2: STANDARD THERMOCOUPLE REFERENCE DESIGNATIONS

In most applications, the length and gauge of the thermocouple wire do not affect the accuracy of the measurement. Due to the high input impedance and lack of dynamic switching in the signal conditioning circuitry of the EX10xxA, the resistance and capacitance of the thermocouple wire are normally not important factors. If, however, maximum system accuracy is desired, the resistance of the thermocouple wire must be considered as a system error source. As an example, 650 ft of 24 gauge type T wire has a resistance of about 500 Ω. Against the EX10xxA's typical input bias current of 7.5 nA, this creates a voltage error of:

$$500\,\Omega \times 7.5\,\text{nA} = 3.75\,\mu\text{V}$$

To convert this error to its representative temperature error, it is then divided by the slope of the thermocouple characteristic at the temperature of interest. For example, the slope of the type T characteristic at 0 ºC is 39 µV/ºC. The error at this point is then:

$$3.75\,\mu\text{V} \div 39\,\mu\text{V}/^{\circ}\text{C} = 0.1^{\circ}\text{C}$$

This example demonstrates how to evaluate the potential error that a specific wire installation represents. The user is encouraged to evaluate each individual application to ensure that the error is within acceptable bounds. In general, best results will be achieved with the shortest and largest wire that the physical requirements of the application can support.

The EX10xxA offers excellent noise rejection through its high common mode rejection and selectable bandwidth limiting, which allows for high integrity, noise-free measurements with even less-than-ideal wiring setups. However, some common instrumentation wiring practices can be used to achieve or ensure maximum performance.

Shielded thermocouple wire can be used to raise the system's rejection of electrical interference. Shielded wire encloses the two thermocouple wires with a low impedance conductor that should be terminated by the user to a convenient earth ground. The EX10xxA provides an external ground stud that can be used for this purpose, but any earth ground point is acceptable.

Magnetic interference, which is present wherever high currents are flowing, is conversely decreased by minimizing the loop area represented by the two thermocouple wires. That is, the wires should be run closely together from the thermocouple junction to the connections in the thermocouple jack. Fortunately, most thermocouple wire comes with a sheath that covers the two thermocouple conductors, inherently creating a small loop area.

Many test applications involve the monitoring of a test article in a chamber, requiring the routing of numerous types of signals through the chamber's cable access ports. It is recommended that the thermocouple wires be run through a separate port and conduit from cables carrying power or high frequency signals.

Voltage Connections

The EX1000A, EX1016A and EX1032A have 50-pin D-sub receptacles for the voltage input channels. These channels are differential inputs and are mapped as pairs (see Table 2-3). The center row of each 50-pin D-sub is connected to chassis GND. This allows for optimized shield termination when using shielded pair wiring.

P1 Connector	
Pin	**Signal**
1	CJC0_RETURN
2	CH07-HI
3	CH07-LO
4	CH06-HI
5	CH06-LO
6	CH05-HI
7	CH05-LO
8	CH04-HI
9	CH04-LO
10	CH03-HI
11	CH03-LO
12	CH02-HI
13	CH02-LO
14	CH01-HI
15	CH01-LO
16	CH00-HI
17	CH00-LO
34	CJC0_INPUT
35	CH15-LO
36	CH15-HI
37	CH14-LO
38	CH14-HI
39	CH13-LO
40	CH13-HI
41	CH12-LO
42	CH12-HI

P2 Connector	
Pin	**Signal**
1	CJC4_RETURN
2	CH23-HI
3	CH23-LO
4	CH22-HI
5	CH22-LO
6	CH21-HI
7	CH21-LO
8	CH20-HI
9	CH20-LO
10	CH19-HI
11	CH19-LO
12	CH18-HI
13	CH18-LO
14	CH17-HI
15	CH17-LO
16	CH16-HI
17	CH16-LO
34	CJC4_INPUT
35	CH31-LO
36	CH31-HI
37	CH30-LO
38	CH30-HI
39	CH29-LO
40	CH29-HI
41	CH28-LO
42	CH28-HI

P3 Connector	
Pin	**Signal**
1	CJC8_RETURN
2	CH39-HI
3	CH39-LO
4	CH38-HI
5	CH38-LO
6	CH37-HI
7	CH37-LO
8	CH36-HI
9	CH36-LO
10	CH35-HI
11	CH35-LO
12	CH34-HI
13	CH34-LO
14	CH33-HI
15	CH33-LO
16	CH32-HI
17	CH32-LO
34	CJC8_INPUT
35	CH47-LO
36	CH47-HI
37	CH46-LO
38	CH46-HI
39	CH45-LO
40	CH45-HI
41	CH44-LO
42	CH44-HI

43	CH11-LO	**43**	CH27-LO	**43**	CH43-LO
44	CH11-HI	**44**	CH27-HI	**44**	CH43-HI
45	CH10-LO	**45**	CH26-LO	**45**	CH42-LO
46	CH10-HI	**46**	CH26-HI	**46**	CH42-HI
47	CH09-LO	**47**	CH25-LO	**47**	CH41-LO
48	CH09-HI	**48**	CH25-HI	**48**	CH41-HI
49	CH08-LO	**49**	CH24-LO	**49**	CH40-LO
50	CH08-HI	**50**	CH24-HI	**50**	CH40-HI

TABLE 2-3: VOLTAGE CONNECTOR SIGNAL ASSIGNMENTS

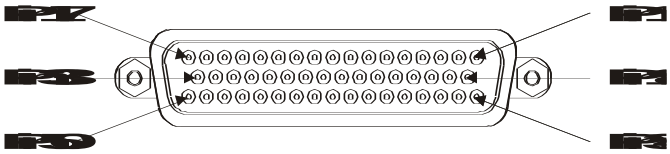


FIGURE 2-13: VOLTAGE CONNECTOR PIN LOCATIONS

Pins 1 and 34 are used for connecting a remote thermistor to the internal signal conditioning circuits. This is used when an external cold plate is being used for temperature measurement. The recommended thermistor is U.S. Sensor P/N: PS103J2 or PS103J2-RoHS. This device is 10 kΩ at 25 °C with ±0.1 ºC initial accuracy and the 0 °C to 50 °C BETA is 3890. The thermistor path, within the EX10xxA product, is single ended and designed to be connected to an electrically isolated thermistor. The use of a grounded thermistor is not recommended.

NOTE Pins 18 through 33 on P1, P2, and P3 are chassis ground pins.

The mating connector for the EX10xxA is a 50-pin D-sub male connector, and is available from the following companies:

Manufacturer	Part Number	Description
Conec	163X11159X	Connector
Conec	165X13409XE	Backshell
NorComp	171-050-103L001	Connector
NorComp	977-050-020R121	Backshell

This is only a partial listing as there are many sources for D-sub connectors.

The EX1000A-51C have 50-pin D-sub receptacles for the voltage input channels. These channels are differential inputs and are mapped as pairs (see Table 2-34).

P1 Connector		P2 Connector		P3 Connector	
Pin	**Signal**	**Pin**	**Signal**	**Pin**	**Signal**
1	CH48-LO	**1**	CH49-LO	**1**	CH50-LO
2	CH07-HI	**2**	CH23-HI	**2**	CH39-HI
3	CH07-LO	**3**	CH23-LO	**3**	CH39-LO
4	CH06-HI	**4**	CH22-HI	**4**	CH38-HI
5	CH06-LO	**5**	CH22-LO	**5**	CH38-LO
6	CH05-HI	**6**	CH21-HI	**6**	CH37-HI
7	CH05-LO	**7**	CH21-LO	**7**	CH37-LO
8	CH04-HI	**8**	CH20-HI	**8**	CH36-HI
9	CH04-LO	**9**	CH20-LO	**9**	CH36-LO
10	CH03-HI	**10**	CH19-HI	**10**	CH35-HI
11	CH03-LO	**11**	CH19-LO	**11**	CH35-LO
12	CH02-HI	**12**	CH18-HI	**12**	CH34-HI
13	CH02-LO	**13**	CH18-LO	**13**	CH34-LO
14	CH01-HI	**14**	CH17-HI	**14**	CH33-HI
15	CH01-LO	**15**	CH17-LO	**15**	CH33-LO
16	CH00-HI	**16**	CH16-HI	**16**	CH32-HI
17	CH00-LO	**17**	CH16-LO	**17**	CH32-LO

34	CH48-HI	**34**	CH49-HI	**34**	CH50-HI
35	CH15-LO	**35**	CH31-LO	**35**	CH47-LO
36	CH15-HI	**36**	CH31-HI	**36**	CH47-HI
37	CH14-LO	**37**	CH30-LO	**37**	CH46-LO
38	CH14-HI	**38**	CH30-HI	**38**	CH46-HI
39	CH13-LO	**39**	CH29-LO	**39**	CH45-LO
40	CH13-HI	**40**	CH29-HI	**40**	CH45-HI
41	CH12-LO	**41**	CH28-LO	**41**	CH44-LO
42	CH12-HI	**42**	CH28-HI	**42**	CH44-HI
43	CH11-LO	**43**	CH27-LO	**43**	CH43-LO
44	CH11-HI	**44**	CH27-HI	**44**	CH43-HI
45	CH10-LO	**45**	CH26-LO	**45**	CH42-LO
46	CH10-HI	**46**	CH26-HI	**46**	CH42-HI
47	CH09-LO	**47**	CH25-LO	**47**	CH41-LO
48	CH09-HI	**48**	CH25-HI	**48**	CH41-HI
49	CH08-LO	**49**	CH24-LO	**49**	CH40-LO
50	CH08-HI	**50**	CH24-HI	**50**	CH40-HI

TABLE 2-4: VOLTAGE CONNECTOR SIGNAL ASSIGNMENTS

The Pin out on the 3 D-sub connectors will provide the connection for additional channels as detailed below:

a) P1 Connector, Pin 1 and Pin 34 becomes channel 49

b) P2 Connector, Pin 1 and Pin 34 becomes channel 50

c) P3 Connector, Pin 1 and Pin 34 becomes channel 51

RX10xx Connections

In RX10xx, the channel wiring is via Screw Terminals and is brought out through a cable gland. Refer to Figure 2-13.

The Power Lines and the other IO Pins are routed through a 13 Pin Circular Connector (P1), refer to Figure 2-14 and Table 2-4.

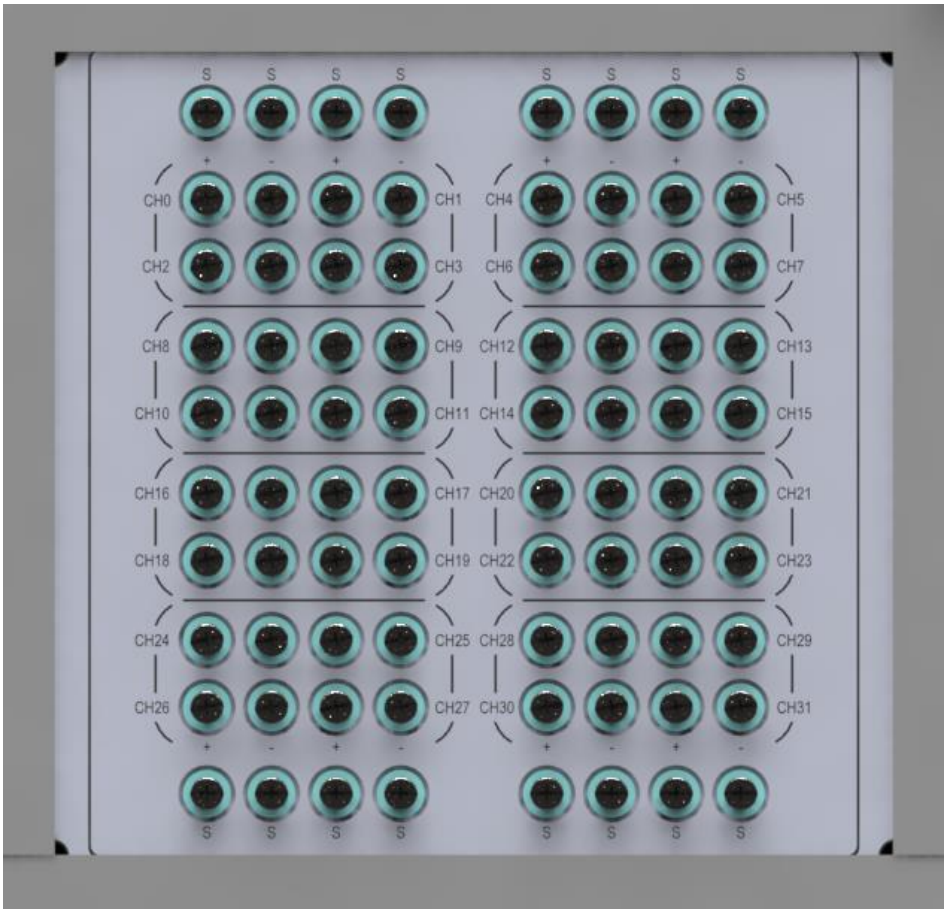


FIGURE 2-13: CHANNEL SCREW TERMINAL PLATE SHOWING 32 CHANNELS ON RX10XX

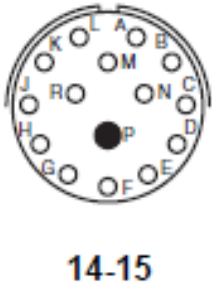


FIGURE 2-14: 15 PIN CIRCULAR CONNECTOR ON RX10XX (P1)

Pin	Description	Remarks
P1/R	+28V_IN	+28V DC Power Input
P1/P	28V_RET	28V Return DC Power Input
P1/P	28V_RET	28V Return DC Power Input
P1/N	+28V_IN	+28V DC Power Input
P1/A	DMM_HI_OUT	*Not to be Used (Factory Calibration Purpose)*

P1/L	**DMM_LO_OUT**	***Not to be Used (Factory Calibration Purpose)***
P1/B	**DAC0_OUT1**	***Not to be Used (Future Purpose)***
P1/K	**DAC0_OUT2**	***Not to be Used (Future Purpose)***
P1/C	**DAC1_OUT1**	***Not to be Used (Future Purpose)***
P1/J	**DAC1_OUT2**	***Not to be Used (Future Purpose)***
P1/M	**AGND**	***Not to be Used (Future Purpose)***
P1/D	**DIO1**	**General Purpose Digital I/O**
P1/H	**DIO2**	**General Purpose Digital I/O**
P1/E	**DIO3**	**General Purpose Digital I/O**
P1/G	**DIO4**	**General Purpose Digital I/O**
P1/F	**DGND**	**GND for General Purpose DIO**

TABLE 2-5: 13 PIN CIRCULAR CONNECTOR SIGNAL ASSIGNMENTS IN RX10XX (P1)

NOTE: 22 AWG wire can be used for the P1 connector cable looming



ATTENTION - Important safety instructions for RX10xx input supply

THE FOLLOWING PIN CONNECTION TO BE CAREFULLY CONNECTED WITH EXTERNAL SUPPLY. ANY MISTAKE IN THE CONNECTION MAY LEAD TO SYSTEM DAMAGE

Pin	Description
P1/R	**+28V_IN**
P1/P	**28V_RET**
P1/P	**28V_RET**
P1/N	**+28V_IN**

SECTION 3

BASIC OPERATION

INTRODUCTION

This section expands on the description of the EX10xxA's / RX10xx's features and explains how to best use them.

ENGINEERING UNIT (EU) CONVERSION

Each EX10xxA/RX10xx input channel can be individually configured for one of eleven different EU conversions. The selections and their definitions are:

Conversion	Parameter
Voltage	mV
Type J	J
Type K	K
Type T	T
Type E	E
Type S	S
Type R	R
Type B	B
Type N	N
User-defined 0	User0
User-defined 1	User1

TABLE 3-1: ENGINEERING UNIT CONVERSION SETTINGS

Voltage The instrument will return the raw voltage measured at its input with units of volts (V). It is unaffected by the measured or input CJC temperature for that channel.

Type J, K, T, E, S, R, B, N The instrument will return the compensated thermocouple temperature measured at its input with units of temperature (ºC or ºF). The thermocouple calculations are performed using the full-order polynomial equations and coefficients from the NIST ITS-90 Thermocouple Database.

User-defined 0, User-defined 1 The instrument will return the compensated thermocouple temperature measured at its input with units of temperature (ºC or ºF). The thermocouple calculations are performed using user-defined coefficients for the polynomial equations. More information on this is given under *User-defined Conversions*.

The default selection is voltage.

NOTE	In a mixed thermocouple system, it is very easy to accidentally mismatch the hardware setup and the software configuration setup. Care is especially warranted, as the resultant errors may not be large enough to obviously indicate a problem, but be significantly larger than the accuracy specification of the instrument.

Linear Correction

The EX10xxA provides the capability to apply a linear transformation on all measurement data after EU conversion. Each channel can be programmed with a gain and offset which are applied as gain*(data-offset).

Gains default to 1.0 and offsets default to 0.0.

VOLTAGE MEASUREMENT RANGES

The EX10xxA provides ±0.067 V and ±0.1 V voltage measurement ranges on thermocouple channels and voltage channels additionally provide ranges of ±0.01 V, ±1 V, and ±10 V. Voltage measurement range is programmable on a per-channel basis.

EX10xxA-51C has only option for voltage channels in the following range ±0.01 V, ±1 V, and ±10 V. Voltage measurement range is programmable on a per-channel basis.

The default selection for voltage channels is the ±10 V range, while thermocouple channels default to a range of ±67 mV.

HARDWARE FILTER

Each EX10xxA input channel can be individually configured with a hardware filter of 4 Hz, 15 Hz, 40 Hz, 100 Hz, 500 Hz, and 1 kHz cutoff frequency. Of the five settings, the 4 Hz setting is suitable for the majority of temperature measurement applications. It offers sufficient response to pass medium gauge thermocouple signals without distortion, while providing the highest degree of (50/60) Hz common mode rejection. This setting also provides the lowest noise floor and is recommended for all applications that do not require a higher bandwidth. For those that do, however, the additional filter setting provides that capability. Their response will pass the output from even the fastest fine-gauge thermocouples with little distortion.

The default selection is 4 Hz.

MEASUREMENT RANGE / INPUT PROTECTION

The specified input voltage range of the EX10xxA thermocouple channels is ±67 mV or ±100 mV for voltage measurements. This level refers to the maximum differential voltage, or voltage that is applied between the (+) and (–) input terminals, that can be measured without distortion. The maximum common mode voltage, or voltage that is applied to the (+) and (–) inputs together, that can be applied without causing out-of-specification distortion of the differential measurement is ±10 V. Application of voltage beyond these levels will result in incorrect measurements, but no instrument damage, up to a maximum of ±35 V.

NOTE	The application of voltages beyond ±35 V can permanently damage the EX10xxA.

The measurement range of the EX10xxA in terms of temperature is a function of its input voltage range and the capabilities of the thermocouple sensors themselves. Specifically, the measurement range of the EX10xxA for the standard thermocouple types is the following:

Type	Min (°C)	Max (°C)	Min (°F)	Max (°F)
J	-200	1200	-328	2192
K	-200	1372	-328	2502
T	-200	400	-328	752
E	-200	900	-328	1652
S	-50	1768	-58	3214
R	-50	1768	-58	3214
B	250	1820	482	3308
N	-200	1300	-328	2372

TABLE 3-2: EX10XXA MEASUREMENT RANGE

In addition to the standard thermocouple types, the EX10xxA can accept and measure a custom thermocouple of any type. The effective measurement range in temperature is subsequently determined by the input voltage range of the EX10xxA and the thermocouple transfer function of the custom thermocouple.

In terms of voltage, the thermocouple connectors may only accept an input voltage of ±67 mV when performing temperature conversion. When in voltage mode, these connectors can accept either ±0.067 V or ±0.1 V inputs. The voltage connectors may use either ±0.01 V, ±0.067 V, ±0.1 V, ±1.0 V, or ±10.0 V inputs for both EU temperature conversion or voltage measurements. When temperature is selected, the instrument automatically switches the selected channel(s) to the ±67 mV range. The EX1000A-TC is a special case where thermocouple inputs are used for voltage measurement and may utilize all input ranges available to the voltage connectors. Table 1- provides an overview of each models capabilities.

COLD JUNCTION COMPENSATION (CJC)

For highest accuracy and stability, the EX1000A-TC, EX1016A, EX1032A, EX1048A and RX1032 provide embedded isothermal input sections that are monitored by twelve precision thermistors (based on Model#, number of thermistors may vary), one for every four thermocouple channels. (Note, the EX1000A may utilize external thermistors as described in the *Voltage Connections* section). To ensure that CJC information is current and time correlated with the input channels, CJC channels are measured with every scan, providing a maximum time separation of less than 4 ms between the input channel measurement and its associated CJC measurement. Table 3-3 shows the association between the CJC and input channels.

CJC #	Channel	Input Channel
CJC0	48	0-3
CJC1	49	4-7
CJC2	50	8-11
CJC3	51	12-15
CJC4	52	16-19
CJC5	53	20-23
CJC6	54	24-27
CJC7	55	28-31
CJC8	56	32-35
CJC9	57	36-39
CJC10	58	40-43
CJC11	59	44-47

TABLE 3-3: CJC CHANNEL / INPUT CHANNEL RELATIONSHIP

The user has configuration control over the reporting of the measured CJC data. This control only affects the display of their data, not the actual measurement of them. They are updated with every scan, regardless of their reporting status. If they are reported, their values are provided with units of ºC, unaffected by the °C/°F configuration setting of the input channels.

The EX10xxA also accommodates the use of an external cold junction that is maintained and measured by the user. In this application, the cold junction temperature in ºC is entered into the EX10xxA and enabled on a per channel basis. That is, the use of internal and user-defined CJC inputs can be mixed throughout the unit. This control is unaffected by the °C/°F configuration setting of the input channels.

The default selections are:

- CJC reporting is disabled
- User-defined CJC temperatures are 0.0
- User-defined CJC temperatures are disabled for all channels

TEMPERATURE UNITS

The EX10xxA can output its temperature data with units of ºC or ºF. This is controlled on a global basis, such that all input channels are configured with one setting. This selection applies to input channel data only. CJC measurement data and the input values for external CJC temperatures are fixed at ºC. Similarly, input channels configured for an EU conversion of voltage are unaffected by this setting.

The default selection is °C.

SAMPLING RATE / NOISE PERFORMANCE

The EX10xxA can be configured for a sampling rate up to a maximum of 1 kSa/s, regardless of the number of channels included in the scan list. The selected sampling rate refers to the frequency at which the entire scan list is measured. As such, all channels are measured at the same sampling rate. When the requested sampling rate is significantly less than 1 kSa/s, however, the EX10xxA automatically takes and averages multiple samples. This offers improved noise performance, while maintaining the requested data output rate. For example, 4096 readings of a K type thermocouple at 25 ºC are displayed with the following typical noise profiles at different selected sampling rates:

Sampling Rate*	Noise (ºCp-p)
≤200 Hz	0.01
500 Hz	0.19
1 kHz	0.37
**4 Hz filter applied*	

TABLE 3-4: TYPE T THERMOCOUPLE NOISE PROFILE VS. SAMPLING RATE

As implied by Table 3-4, the maximum averaging occurs once the sampling rate reaches 200 Hz and represents the ultimate noise floor of the instrument. Decreasing the sampling rate further does not result in a lower noise profile. In practice, the noise performance delivered by the EX10xxA is a function of the slope of the thermocouple characteristic for the sensor being used. That is, type E thermocouples (59 µV/ºC) will deliver inherently quieter measurements than type K thermocouples (39 µV/ºC). However, the relative relationship between noise and sampling rate shown above will be true for any thermocouple type. Consequently, for best instrument noise performance, the sampling rate should be set as low as the data collection requirements allow.

NOTE The described averaging mechanism occurs only if the trigger model is configured with a TRIG source of Timer or LXI alarm. For more details, see *Maximizing Measurement Performance*.

For voltage channels, similar results are seen. After taking 512 readings at each input voltage at varying frequencies, the following typical noise profile can be seen.

Scan Rate	±10.0 V	±1.0 V	±0.1 V	±0.067 V	±0.010 V
200 Hz	0.0005	0.00005	0.000005	0.0000035	0.000001
500 Hz	0.0009	0.00009	0.000009	0.000006	0.0000012
1 kHz	0.0019	0.00019	0.000019	0.000013	0.0000019

TABLE 3-5: VOLTAGE MEASUREMENT NOISE PROFILE VS. SAMPLING RATE

SCAN LIST CONFIGURATION

The EX10xxA can be configured to include from 1 to all 48 of its input channels in the scan list and EX10xxA-51C can be configured to include from 1 to all 51 of its input channels in the scan list. Because of the channel independence present in the EX10xxA/EX10xxA-51C design, there are no accuracy, noise, or speed ramifications from the structure of the scan list. Its channel entries can consequently be solely dictated by the user's application requirements. A valid scan list consists of:

- at least one channel
- no more than 48 channels for EX10xxA
- no more than 51 channels for EX10xxA-51C
- no repeated channels

A scan list entered via the embedded web page interface will be scanned in sequential order from the lowest to the highest numbered enabled channel. For greater control, channels can be added to a scan list in any order (although the channels will be scanned in numeric order once the scan list is executed).

Each EX10xxA input channel maintains its high input impedance and operational independence from the other channels regardless of its inclusion in the scan list. That is, the connection or lack of connection to a valid input signal of unscanned channels makes no difference.

SCAN LIST TIMING

Each scan sequence commences with the sequential measurement of the twelve CJC channels for EX10xxA, regardless of the population of the scan list, the use of external CJC temperatures, or the EU conversions employed on the input channels. The enabled scan list channels are then individually measured. In order to provide the tightest time correlation possible between channel measurements, the scan sequencer cycles through the scan list as fast as possible. Because of the averaging mechanism discussed in *Sampling Rate* / Noise Performance, the exact timing between channels depends on the selected sampling rate. For reference, the interchannel timings for different sampling rates are the following:

Sampling Rate	Interchannel Timing (µs)
1 kHz	16.66666666
500 Hz	33.33333333
400 Hz	41.66666666
300 Hz	55.55555555
≤200 Hz	83.33333333

TABLE 3-6: INTERCHANNEL TIMING VS. SAMPLING RATE

Applications that require maximum time correlation of transient signals will benefit from the following suggestions to configure the unit for maximum speed:

- set a sampling rate of 1 kSa/s
- eliminate unneeded channels from the scan list
- employ the 1 kHz setting of the hardware filter

SELF-CALIBRATION

In order to deliver high measurement accuracy over a wide ambient operating temperature range, the EX10xxA provides the ability to perform an instrument self-calibration. During self-calibration, the input signal conditioning paths are disconnected from the input jacks and connected instead to a calibration bus that is driven by an internal calibration source. Through measurement of the conditioning paths at multiple calibration source points, software compensation for circuitry drift since the last full calibration is conducted. Once self-calibration is performed, the *Thermocouple Accuracy* in Table 1-2 is valid for 7 days and over a ±5 ºC ambient temperature change. Note, however, that the unit must still undergo a full calibration at least once a year, regardless of the use of self-calibration.

Self-calibration should be conducted as often as practical, especially if the ambient environment has changed significantly since the previous calibration. However, fast ambient environmental changes should ideally be followed by a period of thermal stabilization before conducting self-calibration to allow the internal circuitry to stabilize to a new thermal operating condition. The self-calibration process completes quickly and does not require removal of the actual input connections, making it convenient to run often.

Similarly, self-calibration should only be performed after the EX10xxA has been allowed to warm-up from a cold start for at least 60 minutes. To protect the user, the instrument will return a warning if self-calibration is initiated prior to the completion of this warm-up time. However, it is only a warning, and it can be overridden by repeating the calibration function call. An override would be completely acceptable, for example, in cases where a) the unit is already fully warmed-up and is quickly moved from one physical location to another or b) instrument line power is briefly lost due to a facility power outage.

Self-calibration does not overwrite, modify, or take the place of the instrument's nonvolatile calibration constants generated by a full calibration. Instead, it generates an additional set of calibration constants that are applied to the measurement calculation after the full calibration constants. By default, self-calibration data is volatile, meaning that it is not saved through instrument resets or power cycles. This ensures that the instrument always initializes with calibration constants generated by a full calibration. This feature is particularly important when the instrument is being shared among multiple users. Each user is consequently sheltered from the actions of others.

Despite having the ability to conduct self-calibration at any time, there may be user applications that require the use of self-calibration, but demand that it create nonvolatile data. The EX10xxA supports that operation as well. Once self-calibration is performed, the data can be stored to nonvolatile memory through a separate function. Similarly, previously stored self-calibration data can be loaded or cleared from nonvolatile memory.

Self-calibration offers a convenient way to mitigate the effects of time and temperature on the signal conditioning circuitry of the EX10xxA, resulting in significant performance improvement. However, it cannot compensate or correct for cold junction thermal stabilization errors that are created by excessive external sources of heat/cold or a population change of input connectors that is not followed by a thermal equilibrium delay. Similarly, any error created from the resistive drop of very long thermocouple wires is outside of the calibration loop and is not eliminated. For more details, see *Maximizing Measurement Performance*

Self-calibration operations can be performed through the embedded web page or instrument driver interfaces. For web page operations, see *Self-calibration* in Section 4. For programming operations, see *Performing Self-Calibration* in 0.

NOTE	EX1044A/EX1044 system requires hard reboot after self-calibration process so that instrument will be calibrated values gets loaded upon reboot.

NOTE	Self calibration is not applicable for RX1032 since during factory calibration process each instrument will be calibrated in precise manner and calibrated data gets stored in the instrument memory. The instrument has to be sent to factory to perform subsequent calibration after the initial calibration date is expired. *Refer below section for* ***Dynamic Calibration in RX1032***

DYNAMIC CALIBRATION IN RX1032

The dynamic calibration should be conducted to calibrate the measurement drift due to the temperature change. For every measurement scan, the values are adjusted based on the system internal temperature.

- The initial calculated Voltage and Temperature coefficients are stored into the RX1032 system.
- At every measurement the measured values are calibrated dynamically using the stored coefficients.
- Based on the present temperature of the system, the values are calibrated and reported.

OPEN TRANSDUCER DETECTION / LIMITS

The EX10xxA provides two unique sets of programmable limits that are used for open transducer detection as well as general purpose input channel monitoring. These limits, termed limit set 0 and limit set 1, are programmable on a per channel basis. Limit conditions are evaluated with each completed scan and are updated with a maximum latency of 25 ms.

Each limit set has an upper and lower limit value, and limit evaluations can be done against either or both values. The output of limit evaluations is presented in three forms. The operation of the front panel LEDs is tied to the upper and lower limit values of limit set 0. The operation of the digital I/O port can be optionally linked to any combination of the upper and lower limit values of either or both limit sets. Finally, this information is accessible through the instrument driver.

Because of its linkage to the front panel LEDs and the typical use of these LEDs as open transducer detection, limit set 0 has a unique operating feature not present in limit set 1. By default, the values in limit set 0 are set automatically, based on the EU conversion and units selection for each channel. Specifically, the upper and lower limit values are set to the upper and lower values of the EX10xxA measurement range, as specified in Table 3-2. If desired, this automatic operation can be disabled, allowing the manual setting of the limit values. In manual operating mode, the limit values remain constant through EU conversion and unit changes. Limit set 1 operates in manual mode only.

As previously mentioned, limit evaluations can be linked to the operation of the digital I/O port. This is called a DIO Limit Event. For more information, see *Digital I/O* and DIO Limit Events.

The clearing of limit conditions is slightly different for each presentation mechanism. The front panel LEDs always operate in a latch configuration, meaning that a channel's LED remains illuminated even if subsequent channel readings within an acquisition sequence are not out of limit. This behavior is critical to identifying an open thermocouple that is intermittent in its failure. Clearing is done at the beginning of a new acquisition. DIO Limit Events can be programmed to operate in latch or non-latch mode. In either case, they are also cleared at the beginning of a new acquisition. Finally, the instrument driver provides information on the accumulated limit status and is cleared at the beginning of a new acquisition. The default values for each limit set are their maximum values.

DIGITAL I/O AND DIO LIMIT EVENTS

The EX10xxA/EX10xxA-51C/EX1044A/EX1044 features an 8-channel digital I/O port on the rear panel of the instrument. This port can be used as an arm/trigger source, for presentation of limit evaluation information, and as a general purpose output device. The digital I/O connector is a standard DB-9 with the following pin assignment:

Pin	Function
1	DIO Channel 0
2	DIO Channel 1
3	DIO Channel 2
4	DIO Channel 3
5	DIO Channel 4
6	DIO Channel 5
7	DIO Channel 6
8	DIO Channel 7
9	GND

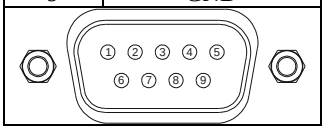


TABLE 3-7: DIGITAL I/O CONNECTOR & PIN ASSIGNMENT

As a general purpose output device, each DIO channel can be independently programmed with regards to its output functionality and its static level to assume when enabled as an output. When not enabled as an output, a channel becomes tri-stated, preventing conflict with other potential voltage drivers. Reference the port's electrical specifications in Table 3-8 for voltage tolerance limits and output drive capabilities. Regardless of output functionality, each channel provides constant input functionality. That is, the input level on each channel can be accessed without a specific enable function call. Moreover, the base functionality of the DIO channels is not affected by triggering, scanning, or any other instrument process. Unless linked to a limit condition, as discussed below, its operation is completely autonomous.

When enabled as an output, each channel also has the ability to generate a 1 µs pulse upon command. An example application of this pulse is to use the EX10xxA to externally trigger another piece of test equipment. The specific operation of the pulse depends on the static level programmed for that channel. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate function call.

For expanded and more automated operation, each DIO channel can be independently linked to one or multiple limit conditions on one or more input channels. This is termed a DIO Limit Event. For example, DIO channel 0 can be programmed to go high when the upper limit of set 0 for channel 2 or the lower limit of set 1 for channel 1 is exceeded. When linked as a limit event, a DIO channel will be cleared at the beginning of a new acquisition. Its state will then be updated with each scan according to the programmed limit evaluations. By default, the cleared state is low, but can be set on a per channel basis to be high through the use of the Invert setting. Similarly, the default operation of each channel is non-latch mode, but can be set on a per channel basis to be latch mode. In latch mode, a transition out of the cleared state would remain, regardless of future limit evaluations, until it is cleared at the beginning of a new acquisition.

It is important to note that the control of the DIO channels through DIO Limit Event assignment does not lock out control through the direct output mechanism. For example, even if a DIO channel has been set high by a DIO Limit Event, it could be asynchronously set low through direct output control. Because limits can only be evaluated as fast as data is being acquired, there could be an application that employs a slow sampling rate but requires the DIO channel to be reset sooner than the normal limit evaluation mechanism would do it. The direct DIO control provides this capability. Use of that control does not alter or disable the limit event mechanism controls; it simply asynchronously alters the level of the DIO channel output. Upon the next scan (and subsequent limit evaluation), the DIO channel will be updated normally per its DIO Limit Event configuration. In general, however, an application will employ only one control mechanism. For that reason, if a DIO channel is linked as a DIO Limit Event, this is noted in the direct control mechanism as a pseudo-warning to guard against accidental use.

The default selections for each DIO channel are:

- output enable is off
- output level is 0

The default selections for DIO Limit Events are:

- all channels are disabled
- invert is off
- latch is off

The electrical specifications for the digital I/O port are provided in Table 3-8. Particular note should be given to the V_{INPUT} specification of -0.5 V to 5.5 V. Exceeding that value with an external voltage source, even though a resistance, could permanently damage the EX10xxA.

Characteristic	Value
V_{INPUT}	-0.5 V to 5.5 V
V_{IH}	2 V min
V_{IL}	0.8 V max
V_{OH} (I_{OH} = -32 mA)	2 V min
V_{OL} (I_{OL} = 64 mA)	0.55 V max

TABLE 3-8: DIGITAL I/O PORT ELECTRICAL SPECIFICATIONS

LXI TRIGGER BUS

The EX10xxA series provides an LXI compatible trigger bus connector set on the rear panel of the instrument. Both connectors have the same pin assignments and are wired in parallel. For more information on the LXI Trigger Bus and creating an appropriate cable, please visit www.lxistandard.org and refer to *LXI Standard Revision 1.2* and *LXI Trigger Bus Cable and Terminator Specifications Rev 1.1*.

Pin	Function	Pin	Function
1	+3.3 V	**14**	RP_TRIG_P0
2	GND	**15**	RP_TRIG_N0
3	RP_TRIG_P1	**16**	RESERVED
4	RP_TRIG_N1	**17**	RP_TRIG_P2
5	GND	**18**	RP_TRIG_N2
6	RP_TRIG_P3	**19**	GND
7	RP_TRIG_N3	**20**	RP_TRIG_P4
8	GND	**21**	RP_TRIG_N4
9	RP_TRIG_P5	**22**	GND
10	RP_TRIG_N5	**23**	RP_TRIG_P6
11	RESERVED	**24**	RP_TRIG_N6
12	RP_TRIG_P7	**25**	RESERVED
13	RP_TRIG_N7		

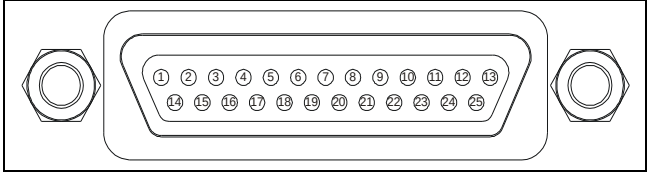


TABLE 3-9: TRIGGER BUS CONNECTOR PIN ASSIGNMENT

For proper operation, each trigger bus connector must be populated with either a cable or a termination. Since the two connectors per unit are in parallel, they are electrically equivalent with respect to receiving the cable or termination. VTI Instruments P/N: 70-0304-000 provides two termination assemblies, which is sufficient to connect any number of instruments, since a

termination is only needed at each end. An example of a three-unit trigger bus installation is shown in Figure 3-1.

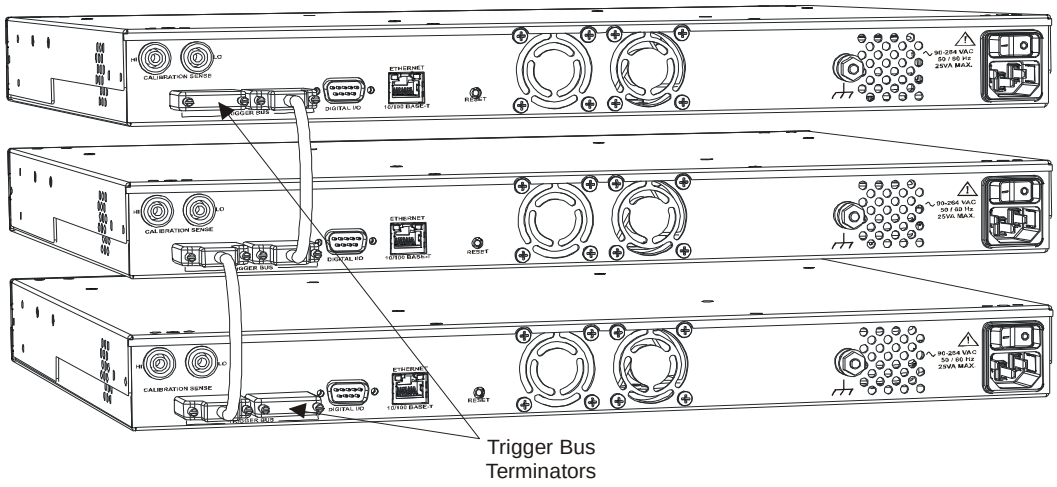


FIGURE 3-1: TRIGGER BUS CABLING EXAMPLE

As a general purpose output device, each VTB channel can be independently programmed with regards to its output functionality and its static level to assume when enabled as an output. When not enabled as an output, a channel becomes tri-stated, preventing conflict with other potential voltage drivers. Regardless of output functionality, each channel provides constant input functionality. That is, the input level on each channel can be accessed without a specific enable function call. Moreover, the base functionality of the VTB channels is not affected by triggering, scanning, or any other instrument process.

When enabled as an output, each channel also has the ability to generate a 1 µs pulse upon command. The specific operation of the pulse depends on the static level programmed for that channel. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate function call.

A common application of the pulse generation is multiple-unit synchronization. Referring to Figure 3-1 above, each unit can be configured to trigger on the same VTB channel. Then one unit is selected as a master and is used to output a signal on the designated VTB channel line, serving as a common trigger source for all three instruments.

The default selections for each VTB channel are:

- output enable is off
- output level is 0

The electrical specifications for the trigger bus are provided in Table 3-10.

Characteristic	Value
Logic Type	M-LVDS Type 2
V_{IT+}	150 mV max
V_{IT-}	50 mV min
V_{OS}	1 V typical

TABLE 3-10: TRIGGER BUS ELECTRICAL SPECIFICATIONS

In addition to a single-master/multiple-slaves configuration, the LXI Trigger Bus can be configured in Wired-OR mode. This mode allows multiple instruments to bring an LXI Trigger Bus channel to a "1" state. The channel will only go back to "0" when all instruments have released it. This is useful

if multiple instruments in parallel operation must all complete before another instrument can begin its operations. Each channel is programmable as to whether it is in Wired-OR or Master-Slave mode.

When an LXI Trigger Bus channel is configured for Wired-OR mode, exactly one instrument on the bus must provide a bias on the channel. This is what allows the channel to fall deterministically back to a "0" state when all other devices release it. Each channel is programmable as to whether it will provide the Wired-OR Bias when it is in Wired-OR mode. default selections:

- Wired-OR mode is off
- Wired-OR Bias mode is off

LOCKING

By default, the EX10xxA allows unrestricted operation from multiple hosts, both through the web page and instrument driver interfaces. While this offers a high level of user flexibility, there are instances where protected operation is desirable, if not required. For these applications, the EX10xxA can be "locked," meaning that it will accept function calls from only the host IP address that issued the lock function. With the EX10xxA in this mode, other host connections that attempt to make calls will be denied.

By design, the locking mechanism is able to be overridden by a secondary host that issues a break lock command. Thus, the lock provides a warning to other users that the unit is in a protected operation state, but not absolute security. This allows for instrument recovery if the locking IP address would become disabled.

Self-calibration requires the acquisition of a lock prior to its initiation.

TRIGGERING

The EX10xxA supports a full function trigger model with a separate arm source and trigger source event structure. For a complete explanation of the trigger model, see **Error! Reference source not found.**. In summary, an acquisition sequence is enabled with a trigger initialize command. Scanning is then initiated upon the receipt of the programmed arm source event followed by the receipt of the programmed trigger source event. Trigger and arm source events can be independently programmed from a variety of sources including Immediate, Timer, Digital I/O, the LXI Trigger Bus, and LXI alarms.

DATA FORMAT

By default, the data returned during data retrieval is limited to the channel readings and the absolute time of scan initiation. If enabled, the EX10xxA can also return the measured CJC temperatures and the delta timestamp of each channel reading from the scan initiation time.

ACQUIRING DATA

In general, the acquisition and retrieval of data on the EX10xxA are conducted with discrete commands that are often separated in time to a large degree. The EX10xxA utilizes a 14 MB FIFO memory storage to buffer acquisition data prior to retrieval. This reading buffer is cleared upon receipt of the trigger initialize command in preparation for reading storage. Then, upon fulfillment of the programmed trigger model conditions, the EX10xxA initiates the scanning of the channel configuration that is defined in the scan list. It continues scanning and storing the acquisition data until the trigger and arm count quantities are reached or the acquisition is aborted. At that point, scanning ceases, and the trigger model is returned to the INIT layer.

The amount of scans that can be buffered within the memory is dependent on the number of channels in the scan list and the requested data format. For this reason, very long scan sequences (in terms of count) will benefit from a minimization of the data format, so as to maximize the number of scans that can be stored. Specifically, the number of scans that can be buffered (Page_Count) is determined by the following formula:

$$\text{Channel_Size} = 4 + 2 \times \text{Timestamp_Reporting}$$

$$\text{CJC_Channel_Size} = 4 + 2 \times \text{Timestamp_Reporting}$$

$$\text{CJC_Channel_Count} = 12 \times \text{CJC_Reporting}$$

$$\text{Page_Size} = 92 + \text{Channel_Count} \times \text{Channel_Size} + \text{CJC_Channel_Count} \times \text{CJC_Channel_Size}$$

$$\text{Page_Count} = 14680064 / \text{Page_Size}$$

where: Channel_Count 1-48 (number of channels in scan list)
CJC_Reporting 0 or 1 (1 = YES, 0 = NO)
Timestamp_Reporting 0 or 1 (1 = YES, 0 = NO)

Page_Count sizes for some typical configurations are the following:

Channel_Count	CJC_Reporting	Timestamp_Reporting	Page_Count
1	0	0	152916
16	0	0	94101
32	0	0	66726
48	0	0	51689
48	1	0	44216
48	0	1	38630
48	1	1	32477

TABLE 3-11: EXAMPLE PAGE COUNT SIZES

If the reading buffer fills to its maximum capacity during scanning, the behavior with regards to additional readings is governed by the setting of the instrument's blocking mode. With blocking mode enabled, additional readings are discarded, leaving the contents of the buffer intact. With blocking mode disabled, the buffer becomes circular, in that additional readings overwrite the oldest readings.

The default selection for blocking mode is disabled.

NOTE The reading buffer memory is volatile and is cleared upon an instrument reset or power cycle.

RETRIEVING DATA

In general, acquisition data is retrieved from the EX10xxA upon the completion of the acquisition. This mode of operation is fully supported by the embedded web page as well as the instrument driver. In addition, data retrieval in the midst of an acquisition sequence is supported by the instrument driver. In either case, retrieved data is fully calibrated and compensated by the EX10xxA and output directly in the requested units. No post-acquisition user manipulations are required. Data can be retrieved one page (scan) at a time or downloaded in multiple page blocks up to the maximum number of pages that exist on the instrument at the time of retrieval. In order to provide the maximum reading buffer capacity for future acquisitions, data is deleted from the FIFO memory upon retrieval.

Applications may retrieve data from this FIFO using either the Read FIFO or Streaming Data interfaces. Please refer to the *Retrieving Data (Read FIFO and Streaming Data)* section for further details. Once data is retrieved from the FIFO, via either method, it is no longer kept within the FIFO.

USER-DEFINED CONVERSIONS

The EX10xxA nominally accepts standard thermocouple types and performs its thermocouple calculations using polynomial coefficients from the NIST ITS-90 Thermocouple Database. In some applications, however, a user may want to override the embedded coefficients with a user-defined coefficient set. One reason to do this is if the transfer function of the specific thermocouple being used has been completely characterized to an accuracy level that exceeds standard thermocouple limits of error. Another reason to do this is if a non-standard thermocouple is used. Up to two unique sets of coefficients can be entered. Specifically, the use of custom thermocouple equations requires the user to know or generate the coefficients for two conversion polynomials.

The *forward conversion polynomial* is used to convert a CJC temperature into a compensating cold junction voltage and has the form of:

$$E = c_0 + c_1 * t^1 + c_2 * t^2 + \ldots + c_{12} * t^{12}$$

where E is in volts, t is in ºC, and c_0 - c_{12} are the coefficients.

The *inverse conversion polynomial* is used to convert a compensated input voltage into temperature and has the form of:

$$t = d_0 + d_1 * E^1 + d_2 * E^2 + \ldots + d_{12} * E^{12}$$

where E is in volts, t is in ºC, and d_0 - d_{12} are the coefficients.

The default values for the coefficients are 0.0.

NOTE	The entry of user-defined coefficients does not automatically enable their use. The enabling is done by setting the EU conversion to User0 or User1.

SECTION 4

TRIGGERING

OVERVIEW

The EX10xxA supports a full function trigger model with a separate arm source and trigger source event structure. The trigger model is based on the industry standard SCPI 1999 Trigger Subsystem and is diagramed in Figure 4-2.

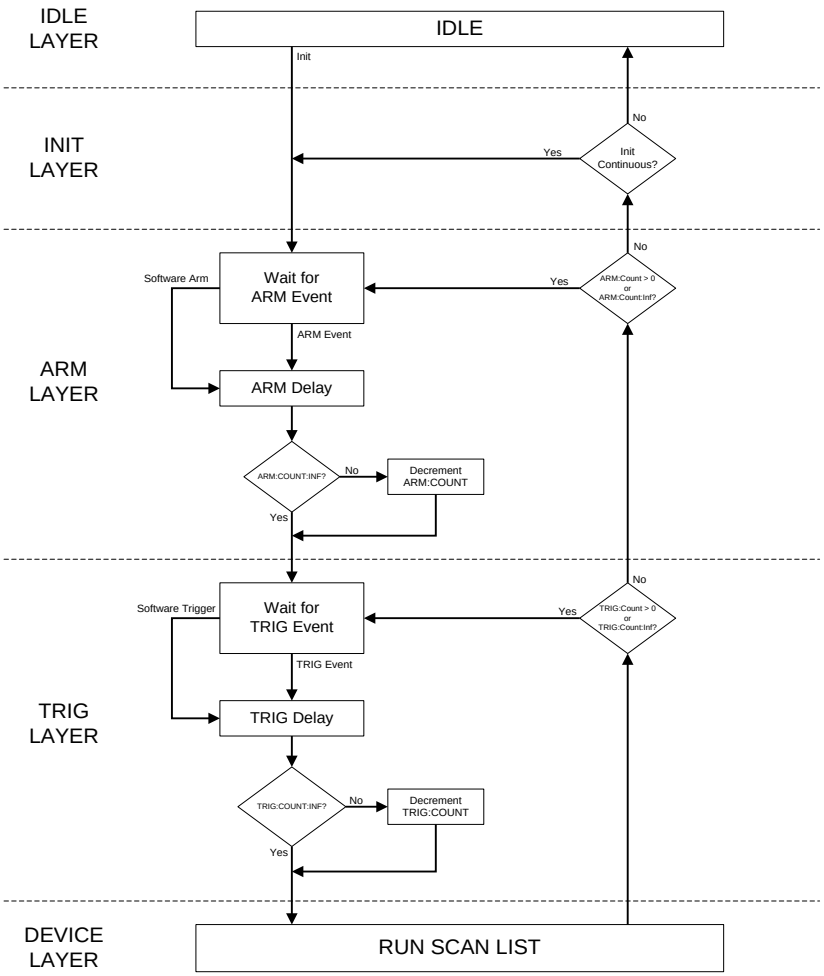


FIGURE 4-2: EX10XXA TRIGGER MODEL

The trigger model is sectioned into five layers: IDLE, INIT, ARM, TRIG, and DEVICE. The EX10xxA reset condition places it in the IDLE state. A trigger initialize command begins the acquisition sequence by transitioning the instrument through the INIT layer into the ARM layer. As this occurs, the reading buffer memory is cleared, and all enabled limit events are reset to their default states.

Upon entering the ARM layer, the ARM Count is reset to its specified value. The instrument remains in the ARM layer until the specified ARM event occurs or a Software Arm is issued. Once that occurs, the specified ARM Delay is waited, the ARM Count is decremented, and the instrument transitions into the TRIG layer.

Upon entering the TRIG layer, the TRIG Count is reset to its specified value. The instrument remains in the TRIG layer until the specified TRIG event occurs or a Software Trigger is issued. Once that occurs, the specified TRIG Delay is waited, the TRIG Count is decremented, and the instrument transitions into the DEVICE layer.

In the DEVICE layer, the scan list is measured, calibrated, and stored into local memory. In addition, limit evaluation is conducted, and enabled limit event states are updated.

If the TRIG Count remains nonzero, the instrument stays in the TRIG layer until the specified TRIG event (and subsequent device action) occurs enough times to decrement it to zero. Once the TRIG Count reaches zero, the instrument then evaluates the remaining ARM Count and repeats the ARM layer action if it is nonzero. However, since each transition into the TRIG layer resets the TRIG Count, each additional ARM layer action results in the full specified number of TRIG Count actions through the TRIG layer and DEVICE layer.

Once the ARM Count reaches zero, the instrument transitions back into the INIT layer. If Init Continuous mode is enabled, the ARM layer is automatically reentered without the issuance of a trigger initialize command. However, unlike with a trigger initialize, enabled limit events are not reset, and the reading buffer memory is not cleared. Conversely, if Init Continuous mode is disabled, the instrument is returned to the IDLE layer and requires the issuance of a new trigger initialize command to begin a new acquisition sequence.

EVENTS

There are two events that control the progress through the trigger model: the ARM event and the TRIG event. Each of these events can be programmed independently to be activated from any combination of the Trigger Bus, Digital I/O port, LAN events, internal system timer, and LXI alarm. In addition, the Trigger Bus, Digital I/O port, and timer can be programmed to be Immediate, creating a permanent satisfaction of the event monitor. Each event monitor can also be bypassed on command with the issuance of a Software Arm or Software Trigger, as appropriate. Software Arms and Software Triggers are always enabled, regardless of the other programmed event sources.

The Digital I/O port and the Trigger Bus monitor the digital hardware ports on the rear panel of the instrument. An event can be controlled by any combination of the eight channels of each port. Moreover, each channel can be independently programmed to monitor a transition edge or a level.

LAN Events are triggered through the eight LAN Event channels via Ethernet packets (UPD/multicast or TCP messages). They are configured in a manner similar to the Digital I/O port and Trigger Bus. These events can occur in “past time”, “now”, or “future time”. “Past time” events have an IEEE 1588 time that occurred in the past. They function essentially as an immediate (or “now” event), but, unlike the “Now” event, provide both sent and received information for debugging purposes. “Now” events have a IEEE 1588 time of “0”, indicating that the event will occur immediately after it is received). Because they have a time of “0”, the event log will only identify when this event was received. “Future time” events have an IEEE 1588 time that has not yet occurred. Because these events occur in the future, they can be prepared for an provide a better response time than either “past” or “now” events. For more information on LAN Events, please see the *LAN Events Menu* in Section 4.

The instrument's internal system timer generates ticks at the specified timer interval, where each tick represents a unique event. For example, a TRIG event source set to Timer at a timer interval of 0.005 s will generate internal triggers at a rate of 200 Hz.

The LXI alarm is specified by *IVI 3.15, IVILxiSync* specification. Similar to the timer, the LXI alarm will cause an event, but these events are based on IEEE 1588 time and always occur at the pre-programmed time. The alarm period can be set for this event as well, allowing the LXI alarm to fire repeatedly at a defined interval.

As previously mentioned, an event condition can be designated to be any combination of Trigger Bus channel transitions or levels, Digital I/O channel transitions or levels, and Timer ticks. If multiple sources for an ARM event are specified, they are logically combined as follows:

ARM event = [(Timer tick event) AND (LXI alarm event) AND (LAN event) AND (Digital I/O event) AND (Trigger Bus event)]

If multiple sources for a TRIG event are specified, they are logically combined as follows:

TRIG event = [(Timer tick event) AND (LXI alarm event) AND (LAN event) AND (Digital I/O event) AND (Trigger Bus event)]

Within each digital hardware port (and LAN events), it is also possible to select multiple channels and multiple conditions for a specific channel. In that case, they are logically combined as follows:

Digital I/O event = (Ch 7 events) AND (Ch 6 events) … AND (Ch 0 events)

Finally, within each channel, the four event conditions of Pos. Edge, Neg. Edge, Pos. Level, and Neg. Level are OR'ed together.

As an example, if a Digital I/O event is specified to be a combination of a Pos. Level on channel 3, a Pos. Edge on channel 6, and a Neg. Edge on channel 6, the event will be satisfied when a Pos. Level on channel 3 occurs simultaneously with a Pos. Edge or a Neg. Edge transition on channel 6. While the Digital I/O event structure was used as an example, the Trigger Bus event structure operates in the same manner.

NOTE	The extensive flexibility of the trigger model system permits the creation of very specialized trigger conditions, which is a powerful application tool. However, it also permits the creation of trigger conditions that would be very difficult to satisfy in practice. For example, if edge conditions are specified on multiple digital hardware channels, the edges must occur within 25 ns of each other to be recognized as having occurred simultaneously. Similarly, the timer source should not be combined with any digital hardware edge conditions.

MAXIMIZING MEASUREMENT PERFORMANCE

As explained in *Sampling Rate* / Noise Performance in Section 3, the EX10xxA takes advantage of a sampling rate setting of less than 1 kSa/s to acquire and average multiple samples. This offers improved noise performance, while maintaining the requested data output rate. However, the trigger model must be configured in a specific manner to allow this to occur. Specifically, the TRIG event source must be configured for Timer or LXI alarm must be enabled and configured. This is critical, because only under this condition does the EX10xxA know exactly the rate and timing of the trigger events. That is, it knows the available time in which to complete the acquisition and consequently how many samples it can average and still complete the entire scan list. With any other trigger source, the triggers can occur at any time. In that case, the EX10xxA completes the scan list in minimum time and does not average.

If the desired trigger model configuration requires the use of the digital I/O port and/or the trigger bus, it is thus best to configure the ARM source for the required digital event, leaving the TRIG source as Timer. The TRIG source should be changed from Timer only in the situation where triggering is required on a two-stage digital event. An example of this is arming on a digital I/O channel level and then triggering on a trigger bus channel edge.

MAXIMUM TRIGGER RATE

The ability to configure the TRIG event source to be an external event such as a Digital I/O channel level or a Trigger Bus channel edge is a powerful test feature, allowing measurements to be precisely correlated with external events. External events, however, could nominally be generated at a rate far faster than the EX10xxA can completely and correctly execute the DEVICE layer. To prevent the instrument from being overdriven, the trigger model contains an internal pacer timer. Once the trigger event has been satisfied, this timer disables the recognition of all trigger events for a time period of 1 ms. Trigger events that occur during this 1 ms period are ignored, not buffered. This is similar in concept to the fact that, for example, additional ARM events are ignored if the trigger model is not specifically waiting for an ARM event. Due to this action, triggering the EX10xxA with a burst of external pulses that exceeds 1 kHz will result in not only fewer readings than trigger events, but also a reading rate that is potentially far less than the maximum rate.

Consider an example in which the trigger model is set to the following configuration:

Init Continuous	Disabled
Timer Interval	0.1
Arm Source	Immediate
Arm Count	1
Arm Delay	0
Trig Source	DIO Channel 0 Positive Edge
Trig Count	1
Trig Delay	0

Once initialized, the EX10xxA is driven by a burst of 100 positive edge transitions at a rate of 1.5 kHz. At 1.5 kHz, the time between trigger events is 990 µs. Since this time is less than 1 ms, every other trigger event will be ignored. Consequently, this burst will generate 50 readings at a rate of 750 Hz. Note that the actual reading output rate is far less than the 1 kSa/s maximum rate. To achieve the maximum output rate, the TRIG event source should be set to Immediate or Timer (with a Timer Interval of 0.001).

SECTION 5

WEB PAGE OPERATION

INTRODUCTION

- LXI Discovery utility (Available at LXI consortium website http://lxistandard.org/Resources/LXIDiscoveryTool.aspx)

To open the embedded web page, simply type the IP address of the instrument into browsers address bar, and navigate accordingly. In order for the browser to communicate with the instrument, the instrument and user computer should be within the same network, and may require permissions from the network/computer administrator.

There are various ways to search the IP address, and other network configuration parameters of the instrument. The easiest method is to refer to the embedded display on the instrument, and use the menu key to navigate to necessary details. However, if the device display is not physically accessible, the user can use LXI Discovery tool, or any standard IO Libraries tool, such as National Instruments (c) Measurement and Automation Explorer (MAX) and the Keysight (formerly Agilent) (c) Connection Expert (ACE) to discover the instrument. The instrument also supports mDNS protocols, so that any tools meant for Zero Configuration Networking, such as Apple ® Bonjour ® or Avahi, can also be used to discover the instrument. For more information on installation of these software utilities, please refer to their respective vendors.

The LXI Discovery tool is a free utility, available for download from LXI Consortium itself. It uses both VXI-11, as well as mDNS protocols to detect the LXI instruments present in your network. Once installed, this software will discover EX10xxA instruments, if available, as shown below.

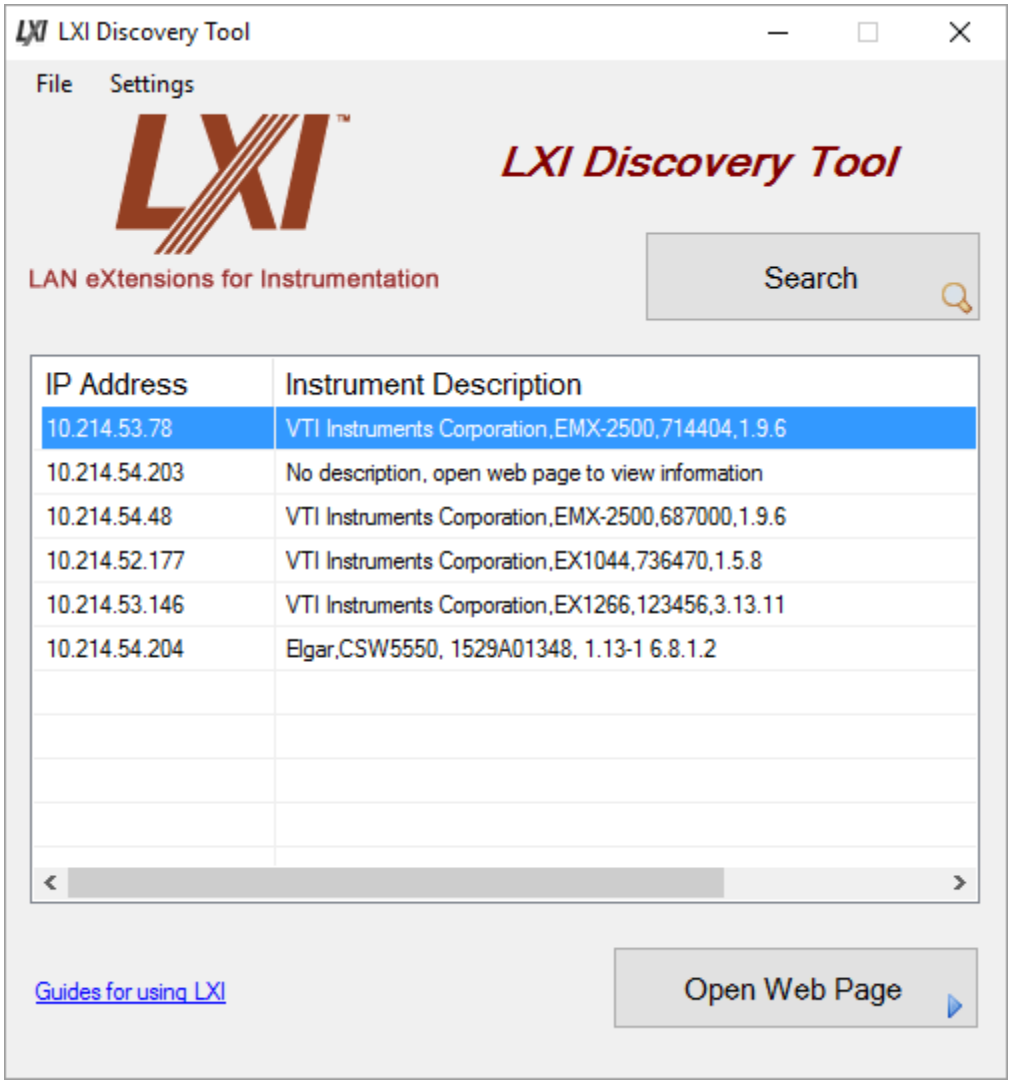


FIGURE 5-1: LXI DISCOVERY TOOL LIST ALL THE INSTRUMENTS

Alternatively, the EX10xxA may also be discovered using Internet Explorer's Bonjour for Windows plug-in. The IP address of the EX10xxA instrument can also be entered into the address bar of any web browser to view the embedded web page. By default, the EX10xxA will first attempt to use DHCP to set its IP Address. If DHCP is not available on the network it is connected to, it will instead use Auto IP. Determining the Auto IP address is discussed in below sections. Other discovery methods, such as VXI-11, can be used as well.

COMMON ISSUES

Although Ethernet is an easy connectivity option, some users may experience difficulty discovering LXI devices. Most often, these issues have been caused by networking issues, such as the instrument and host computer being connected to different sub-nets of the network. The user can contact their system/network administrator for assistance on these issues. Here are few other causes that may be present in the user system which may obstruct instrument discovery:

Multiple Network Ports

It is possible that the user computer, can have multiple network adaptors (physical/virtual). Common example of this case, is use of Wireless Internet/Wi-Fi in Laptop computers, whose Wired Ethernet connections are connected to LXI device connected network. In some cases, virtual network adaptors could have been created for use by some of the installed applications on user system. The easy solution to these problems is to disable all other Ethernet connections and so force the computer to select the network port which is used by LXI Instrument.

Firewall

It is possible that the firewall of the computer may be stopping the data traffic to the connected LXI device. It may not be always feasible to stop the Firewall completely, and hence the user may contact their computer administrator for making exceptions in Firewall configuration to allow LXI data traffic. Here are the list of ports and protocols, typically used by LXI devices.

Protocol / Service	Port	Base Protocol	Remarks
mDNS	5353	UDP/TCP	Zero-config protocol for <hostname>.local address resolution; service discovery; Multicast 224.0.0251, FF02::FB
HTTP	80	TCP	Instrument web pages (SFP) and driver communication
ICMPv4		ICMP	Typically enables echo request/respond for ping
ICMPv6		ICMP	Optional - Typically enables echo request/respond for ping, also SLAAC for IPv6 addresses, RDDNS
Arp		Arp	Used to confirm address assignments
DHCPv4	67/68	UDP/TCP	IPv4 address assignment, DNS Server, Dynamic DNS, Gateway
DNS	53	UDP	Naming service
SSH	22	TCP	Bidirectional interactive text oriented communication
RPC port-mapper	111	UDP/TCP	Builds upon Sun-RPC and port-mapper
VXI-11	Varies	TCP	VXI-11 instrument discovery The exact port number varies; query the RPC port-mapper to determine which port is current in use
LXI-eventsvc	5044	UDP/TCP	LXI Event support for instrument triggering; multicast 224.0.23.159, FF02::138. LXI Events can use other ports also, but default port number is 5044
Ptp-event	319	UDP/TCP	LXI Profile IEEE 1588 Precision Time Protocol (PTP); multicast 224.0.1.129, FF02::181
Ptp-general	320	UDP/TCP	LXI Profile IEEE 1588 Precision Time Protocol (PTP); multicast 224.0.1.129, FF02::181
Data stream	9900	TCP	Driver data stream

TABLE 5-1: STANDARD PORTS, PROTOCOLS, AND SERVICES

GENERAL WEB PAGE OPERATION

When initial connection is made to the EX10xxA, the instrument home page appears. This page displays instrument-specific information including:

- Vendor
- Serial number
- Description
- LXI Information
- Hostname
- MAC Address

- Netmask
- Instrument Address String
- Firmware version
- Date of last full calibration
- Presence of nonvolatile self cal data
- Error status
- IEEE 1588 PTP Time

This page is accessible from any other instrument page by clicking on the instrument name in the web page header.

All of the configuration, data acquisition, and data retrieval features available through the web page are accessed through the EX10xxA command menu that is displayed on the left hand side of every internal web page. The entries on the command menu represent three types of pages:

Status This type of page performs no action and accepts no entries. It provides operational status and information only. The **EX10xxA Main** page is an example of a status page.

Action This type of page initiates a command on the instrument, but does not involve parameter entry. The **Trigger…Initialize** page is an example of an action page.

Entry This type of page displays and accepts changes to the configuration controls of the instrument. The **Scan List…Configuration** page is an example of an entry page.

Use of the entry-type web pages in the EX10xxA are governed by a common set of operational characteristics:

- Each page contains two standard buttons: **Apply**.
- Pages initially load with the currently-entered parameters or selections displayed.
- Changes that are made after the initial load are not accepted until the **Apply** button is clicked. Once the **Apply** button is clicked, the page will reload, showing the new state of the page.
- Navigation through a parameter screen is done with the Tab key. The Enter key has the same function as clicking the **Apply** button and cannot be used for navigation.

When accessing an action or entry page for the first time during a session, an Error page will appear. To clear this error, click on the **login** link. The **Login** page will appear. In the **Password** field, enter **ex1048**.

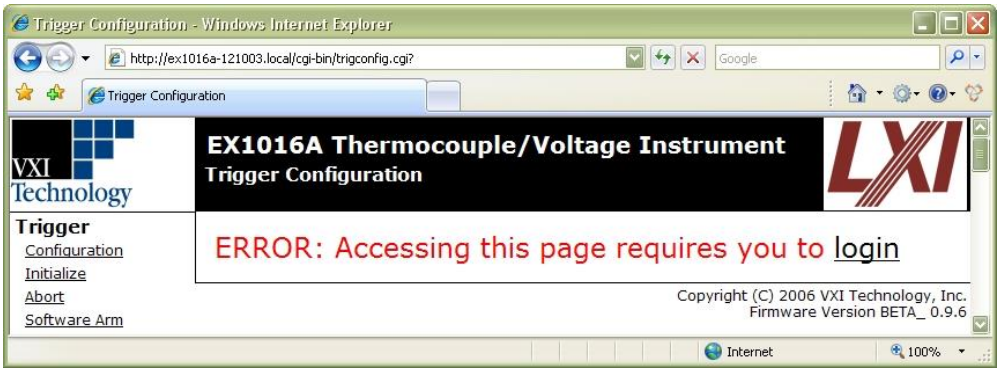


FIGURE 5-2: LOGIN ERROR PAGE

TRIGGER MENU

The **Trigger** menu is used to configure and operate the EX10xxA's trigger system.

Configuration

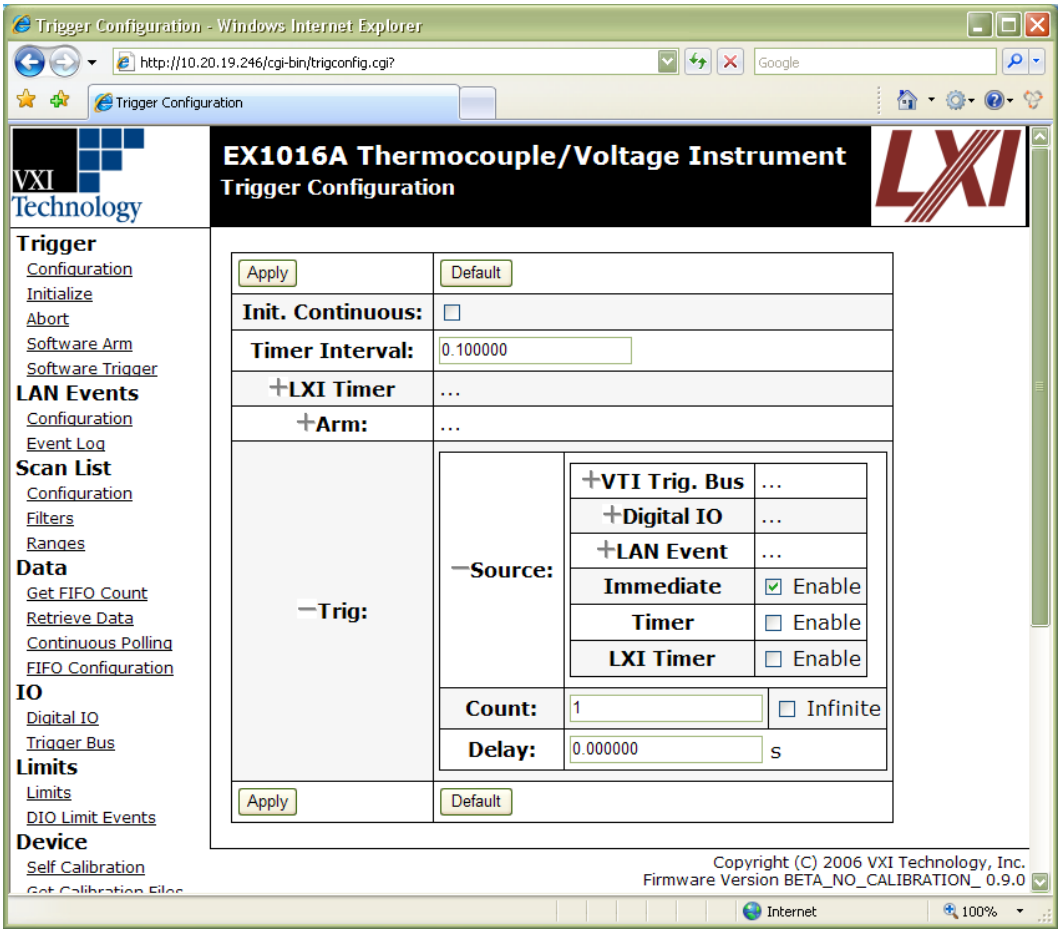


FIGURE 5-3: TRIGGER PAGE

This entry page is used to configure the arm and trigger sources of the EX10xxA. The page is designed with a collapsible and expandable menu structure that is retained between page visits. This allows the user to expose only as much as the trigger system as needed, making navigation through the configuration options easier. Detailed information on triggering is provided in **Error! Reference source not found.**. Most of the controls on this page are checkboxes to enable different levels of functionality. The rest are parameter entry fields that have the following valid input ranges:

Parameter	Min	Max	Resolution
Timer Interval (s)	0.001	4294	0.000001
Arm Count	1	$(2^{31}-1)$	1
Arm Delay (s)	0	4294	0.000001
Trig Count	1	$(2^{31}-1)$	1
Trig Delay (s)	0	4294	0.000001

TABLE 5-2: TRIGGER CONFIGURATION PARAMETER RANGES

In addition to the two standard buttons, this page also provides a **Default** button. This button provides a one-click method of returning all trigger configuration parameters to their reset values.

It should be noted that the use of this button does not eliminate the need to click the **Apply** button to accept the changes.

Initialize

Clicking on this link will begin an acquisition, transitioning the trigger model from the IDLE layer to the ARM layer. This command clears the FIFO reading memory and resets all limit indications.

Abort

Clicking on this link will abort the current acquisition and returns the trigger model to the IDLE layer. Once an acquisition has begun, most commands are not accessible. The abort command is used to prematurely end the acquisition in order to make configuration changes. Acquired data is not affected by the issuance of an abort.

Software Arm

Clicking on this link will perform a software arm of the trigger model. Software arms are always enabled as an ARM source and can be used to perform a bypass of the configured ARM source. The software arm applies at the time it is issued and will be ignored if the system is not waiting for an ARM event. The issuance of a software arm has no effect on the ARM source configuration.

Software Trigger

Clicking on this link will perform a software trigger of the trigger model. Software triggers are always enabled as a TRIG source and can be used to perform a bypass of the configured TRIG source. The software trigger applies at the time it is issued and will be ignored if the system is not waiting for a TRIG event. The issuance of a software trigger has no effect on the TRIG source configuration.

LAN EVENTS MENU

LAN events are used for device communication over Ethernet. The **LAN events** menu is used to configure the EX10xxA's LAN event arm and trigger lines.

Configuration

This entry page is used to define how the EX10xxA responds to LAN events. Each LAN event trigger line can be set to arm or trigger on a packet with the same event ID as specified in the **Event ID** field (default values shown below). Note that each line can arm or trigger on different LAN events (i.e. LAN event line LAN0 could arm on Event ID LAN1, but trigger on Event ID LAN3). Additionally, the user can use the **Filter** field to specify which network hosts are allowed to trigger the LAN event line. The format of this field should be "hostname:port". When the hostname is set to blank or "" (default), the EX10xxA will accept all UDP and TCP packets on port 5044. If set to "ALL", the instrument will allow multicast UDP packets.

From this page, the user can also set the instrument's LAN Domain and configure the LAN event log. By clicking on the **Log Enable** checkbox, the EX10xxA will record LAN events as they occur. The **Log Overflow Mode** radio buttons affect how the log buffer will operate once it becomes full. If **Overwrite Old** is selected, the existing log will be overwritten by new LAN events, while selecting **Ignore New** will ignore all LAN events that occur after the buffer is full.

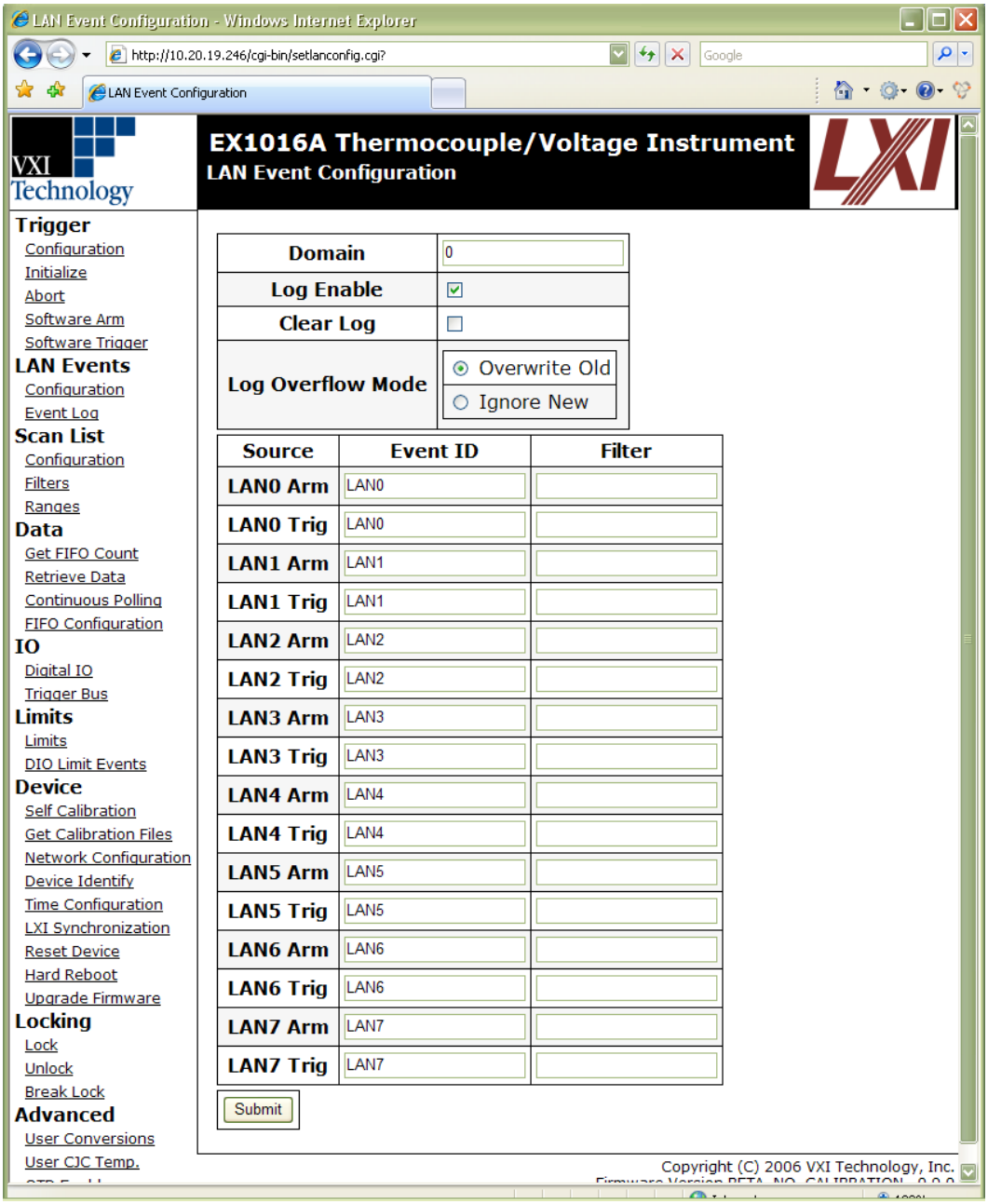


FIGURE 5-3: LAN EVENT CONFIGURATION PAGE

Event Log

This status page allows the user to view the LAN event log, when enabled. The data is provided in a line-by-line manner. Clicking the **Get Next Line** link allows the user to view the next chronological LAN event. The **Clear Text Field** link erases the LAN event that is currently being displayed.

NOTE	Once an event log entry is viewed from this page, it can no longer be accessed through the instrument driver as it is immediately erased after being viewed.

Status

This status page shows the trigger and arm states for all eight LAN event lines. If a “0” is shown in the state field, the associated LAN event line is in a low state. If a “1” is shown, the line is in a high state.

SCAN LIST MENU

The **Scan List** menu is used to configure the EX10xxA input channels with regards to scan list inclusion, EU conversion, and hardware filter setting.

Configuration

This entry page is used to define the channel scan list and the EU conversion setting for each channel. An example of this page is shown Figure 5-4.

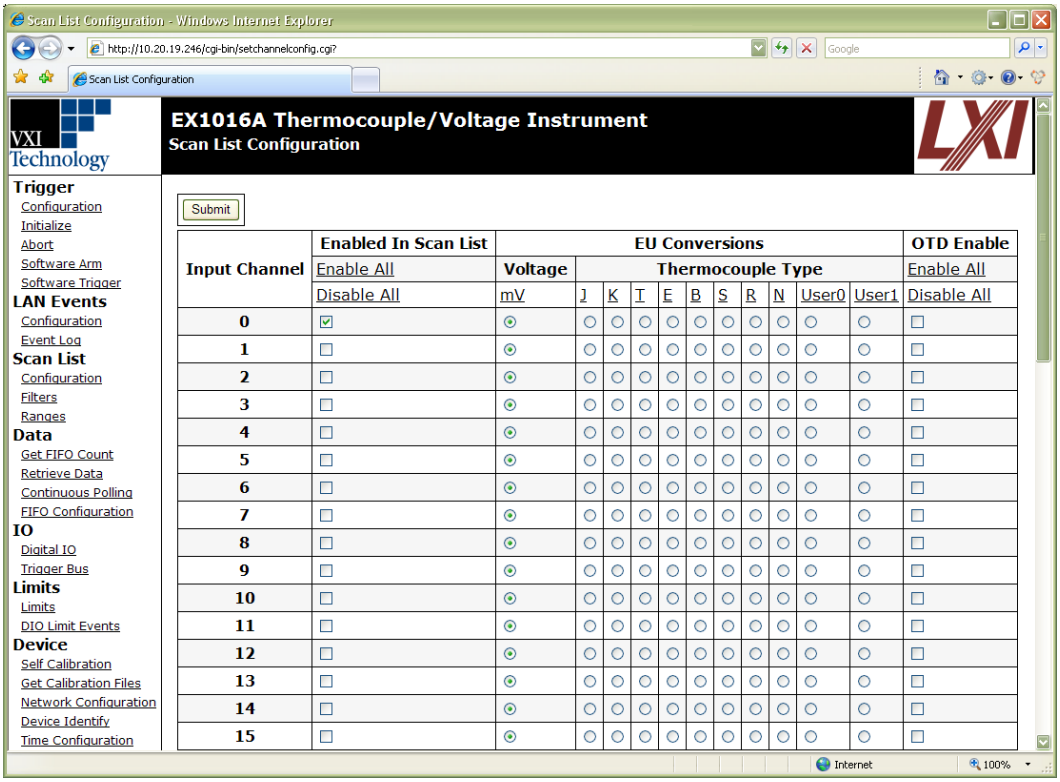


FIGURE 5-4: SCAN LIST CONFIGURATION PAGE

This screen displays that currently channel 0 is enabled in the scan list and is set to perform voltage measurements. Channels are enabled or disabled in the scan list by toggling the appropriate checkbox. Additionally, all channels can simultaneously be enabled or disabled by clicking on the **Enabled** or **Disabled** link. If the channel is a voltage channel, the **Voltage** radio box will be selected. For thermocouple channels, the EU conversion is set by clicking the appropriate radio button under the desired conversion type. Additionally, all channels can simultaneously be set to the same conversion type by clicking on the appropriate link. It should be noted that the use of these links does not eliminate the need to click the **Apply** button to accept the changes. The user can also enable/disable OTD on either a per channel or global basis form by using the OTD Enable check boxes.

Filters

This entry page is used to define the hardware filter setting for each channel. Each channel's hardware filter is set by clicking the appropriate radio button under the desired frequency value. Additionally, all channels can simultaneously be set to the same frequency value by clicking on the appropriate link. It should be noted that the use of these links does not eliminate the need to click the **Apply** button to accept the changes.

Ranges

This entry page is used to define the each channel's input voltage range. Each channel's voltage range is set by clicking the appropriate radio button under the voltage range value. It should be noted that the use of these links does not eliminate the need to click the **Apply** button to accept the changes.

DATA MENU

The **Data** menu is used to retrieve data from the EX10xxA as well as configure the format of the data to be acquired.

Get FIFO Count

This status page displays how many pages (scans) worth of acquisition data have yet to be retrieved.

Retrieve Data

This action page extracts and displays one page (scan) worth of acquisition data on a first-in, first-out (FIFO) basis. If this page is invoked when the FIFO is empty, an error is returned. In order to provide the maximum reading buffer capacity for future acquisitions, data is deleted from the FIFO memory upon retrieval.

Continuous Polling

This action page is used to continuously trigger and retrieve one page (scan) of data from the EX10xxA at a refresh rate of approximately once every three seconds. It is very useful for general data monitoring, installation debugging, and instrument control familiarization. It cannot, however, be used to test the trigger system, as it must alter the trigger configuration settings in order to acquire data. Once continuous polling is initiated, it can be stopped by clicking the **Stop Polling** button or a link to another page. Using the button preserves the most recent data screen.

NOTE	Use of the **Continuous Polling** page alters the trigger configuration settings. Any previously entered configuration settings must be reentered.

FIFO Configuration

This entry page is used to control the data format and overflow behavior of the FIFO memory.

The **Report CJC Temperatures** control enables the reporting of the CJC temperatures in the acquisition data. This control does not affect their measurement, only the display of the data. The CJC temperatures are taken with every scan, regardless of this control. Reported CJC temperatures are always in units of ºC, regardless of the **Temperature Units** selection.

The **Report Timestamps** control enables the reporting of delta timestamps per channel in the acquisition data. The timestamp represents the time, in increments of 100 ns, between the specific channel measurement and the scan initiation time.

The **Temperature Units** selection controls whether the input channel data is returned in units of ºC or ºF. This setting has no effect on the reported CJC data or data for input channels that are configured for an EU conversion of mV.

The **Blocking Mode** control governs the system behavior if the reading buffer fills to its maximum capacity during scanning. With blocking mode enabled, additional readings are discarded, leaving the contents of the buffer intact. With blocking mode disabled, the buffer becomes circular, in that additional readings overwrite the oldest readings.

IO MENU

The **IO** menu is used to receive and output data through the digital I/O port and trigger bus of the EX10xxA.

Digital IO

This entry page is used to monitor the state of the 8-channel digital I/O port and control it as a general purpose output device. Each DIO channel can be independently programmed with regards to its output functionality, its static level to assume when enabled as an output, and pulse operation.

The **Input State** status section displays the measured digital level (0 or 1) of the DIO channels each time the page is accessed or refreshed. The information does not update automatically on an input level transition or on a timer. Consequently, it should only be used to monitor static or slowly changing levels. The input monitoring functionality operates regardless of the state of the **Output Enable** control.

The **Output State** control provides a pull-down selection in which the output level of each DIO channel is specified. This setting represents the level that the channel will assume if/when enabled as an output. When a channel is not enabled, this selection has no effect.

The **Output Enable** control enables each DIO channel as an output. Once enabled, a channel will assume the level specified in the **Output State** control. When not enabled as an output, a channel becomes tri-stated.

NOTE	Before specifying a DIO channel as an output, ensure that no other active voltage drivers are connected to the channel. Otherwise, incorrect operation or instrument damage could result.

The **Pulse** control is used to generate a single-shot 1 µs pulse. The specific operation of the pulse depends on the static level programmed for that channel. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate command. This control is only valid when the channel is enabled as an output. Normally, pulse operation is used once a channel's static output level is already defined. However, if selections are applied together to change the output state and output a pulse, the channel will first assume the new output state and then output the appropriate pulse.

The **Limit Event** status section indicates (Y or N) the linkage of any DIO channels to a limit evaluation condition. When linked, the output state of a channel will be primarily controlled by the limit conditions specified in the **DIO Limit Events** page. The linkage is noted here, because channels linked as DIO Limit Events are normally not used as direct control outputs. However, direct output control is not disabled and can be used to asynchronously reset a linked channel.

Trigger Bus

This entry page is used to monitor the state of the 8-channel trigger bus (VTB) and control it as a general purpose output device. Each VTB channel can be independently programmed with regards to its output functionality, its static level to assume when enabled as an output, and pulse operation.

The **Input State** status section displays the measured digital level (0 or 1) of the VTB channels each time the page is accessed or refreshed. The information does not update automatically on an input level transition or on a timer. Consequently, it should only be used to monitor static or slowly changing levels. The input monitoring functionality operates regardless of the state of the **Output Enable** control.

The **Output State** control provides a pull-down selection in which the output level of each VTB channel is specified. This setting represents the level that the channel will assume if/when enabled as an output. When a channel is not enabled, this selection has no effect.

The **Output Enable** control enables each VTB channel as an output. Once enabled, a channel will assume the level specified in the **Output State** control. When not enabled as an output, a channel becomes tri-stated.

NOTE	Before specifying a VTB channel as an output, ensure that no other active voltage drivers are connected to the channel. Otherwise, incorrect operation or instrument damage could result.

The **Pulse** control is used to generate a single-shot 1 µs pulse. The specific operation of the pulse depends on the static level programmed for that channel. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate command. This control is only valid when the channel is enabled as an output. Normally, pulse operation is used once a channel's static output level is already defined. However, if selections are applied together to change the output state and output a pulse, the channel will first assume the new output state and then output the appropriate pulse.

LIMITS MENU

The Limits menu is used to configure the limit condition evaluation system of the EX10xxA.

Limits

This entry page is used to specify the values for the two programmable limit sets. These limits, termed limit set 0 and limit set 1, are programmable on a per channel basis. Only one set of limits is displayed at a time on the page. Navigation between the sets is provided by the **Set** control, a pull-down selection.

By default, the values in limit set 0 are set automatically, based on the EU conversion and units selection for each channel. Specifically, the upper and lower limit values are set to the upper and lower values of the EX10xxA measurement range, as specified in Table 3-2. For example, if input channel 0 is configured for an E type conversion and a units selection of ºC, its upper and lower limit values will be +900 and -200, respectively. To override the automatic setting and set the values manually, check the **Manual** checkbox and enter new values. Values can be entered in decimal or scientific notation, but will always be displayed in scientific notation. Limit set 1 operates in manual mode only.

NOTE	Limit sets operating in manual mode do not automatically adjust for changes in EU conversion or units selection. For proper operation, limit values should be set after EU conversion and units selections are made and reset upon subsequent configuration changes.

DIO Limit Events

This entry page is used to link one or multiple limit conditions to the operation of a channel on the digital I/O port. Nominally, the results of limit set 0 evaluations are represented in the front panel LEDs, and the results of both limit set evaluations are accessible through the instrument driver. The DIO limit event mechanism offers the additional ability to precisely control a DIO channel through the limit evaluations on one or more input channels.

DIO limit events are programmable on a per channel basis; that is, each of the 8 DIO channels can be configured with a unique set of operational characteristics. Only one DIO channel's limit event controls is displayed at a time on the page. Navigation between the DIO channels is provided by the **DIO Channel** control, a pull-down selection.

The linkage of a DIO channel to a specific limit condition on a specific input channel is done by checking the appropriate checkbox. Multiple selections are allowed and are logically OR'ed together. For example, Figure 5-5 shows that DIO channel 0 is linked to limit conditions on input channels 0, 3, and 5. Specifically, DIO channel 0 will transition if any of the four specified limits is exceeded.

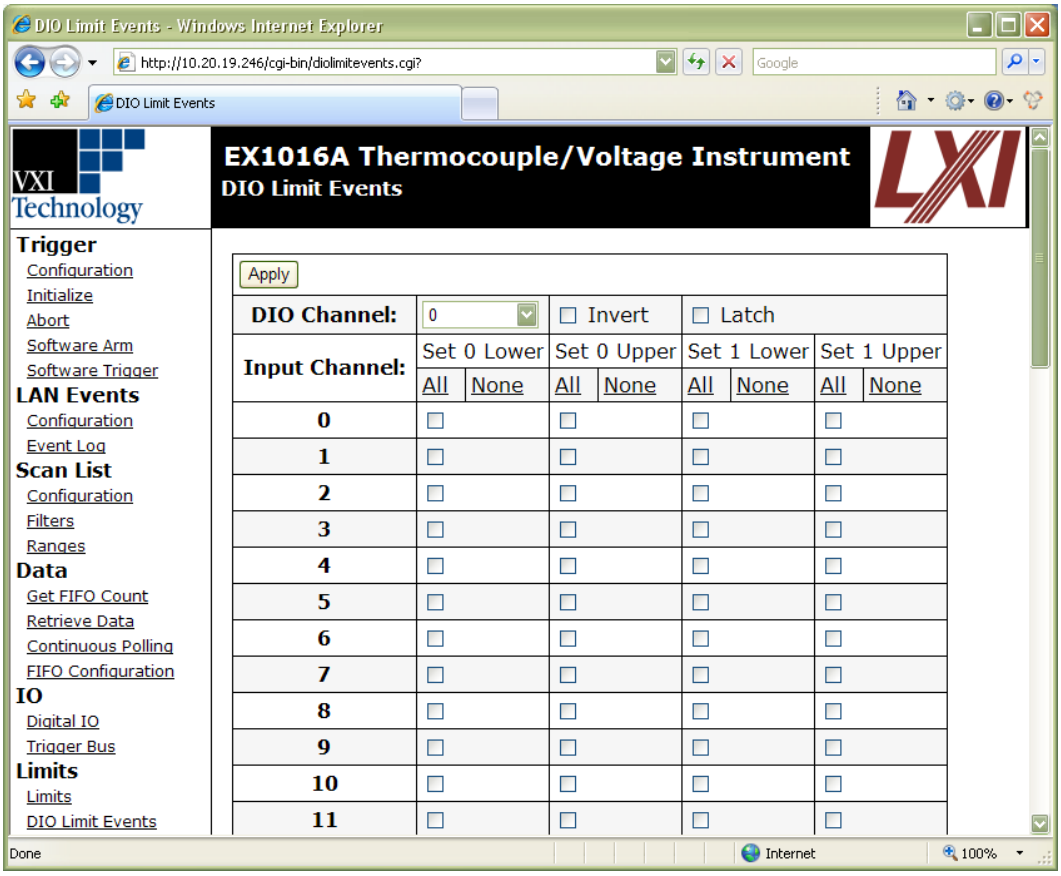


FIGURE 5-5: DIO LIMIT EVENTS PAGE

To ease data entry in complex configurations, the **All** and **None** links can be used to enable or disable a limit set value across all input channels. It should be noted that the use of these links does not eliminate the need to click the **Apply** button to accept the changes.

The DIO limit events and the scan list configuration controls are autonomous. That is, specifying a limit condition in **DIO Limit Events** does not automatically enable it in the scan list. Similarly, removing an input channel from the scan list does not automatically remove it as a linked limit condition. DIO limit event linkages to unscanned channels do not cause an error condition and are simply ignored in the evaluation.

When linked as a limit event, a DIO channel will be cleared at the beginning of a new acquisition. Its state will then be updated with each scan according to the programmed limit evaluations. By default, the cleared state is low, but can be set on a per channel basis to be high through the selection of the **Invert** checkbox. Similarly, the default operation for each channel is non-latch mode, but can be set on a per channel basis to be latch mode through the selection of the **Latch** checkbox. In latch mode, a transition out of the cleared state would remain, regardless of future limit evaluations, until it is cleared at the beginning of a new acquisition.

DEVICE MENU

The Device menu is used to perform advanced configuration operations on the EX10xxA/E.

Self-calibration

This action page is used to perform operations related to self-calibration. As discussed in Self-calibration in Section 3, the measurement performance of the EX10xxA is significantly improved through the periodic use of self-calibration. The self-calibration process completes quickly and does not require removal of the actual input connections or the use of external equipment, making it convenient to run often. In order to take full advantage of the self-calibration system, it is important to understand how the self-calibration constants are generated, used, and stored.

Self-calibration does not overwrite, modify, or take the place of the instrument's nonvolatile calibration constants generated by a full calibration. Instead, it generates an additional set of calibration constants that are applied to the measurement calculation after the full calibration constants. When self-calibration is performed, it places its calculated data in volatile memory. This data is termed the "current self cal data" and is used automatically by the instrument. No additional command is necessary. Because it is in volatile memory, however, it will be cleared upon an instrument reset or power cycle. There is the ability, however, to store the current self cal data into nonvolatile memory.

In order to perform a self-calibration, a lock on the instrument must first be acquired. For more information on locking, refer to the **Locking Menu** description in this section. If self-calibration is attempted without the acquisition of a lock, an error message will be displayed. Within this page, the self-calibration process is initiated by clicking on the **Perform Self Calibration** button. Once started, the page will confirm the initiation of the calibration process and then continuously refresh with a percentage completion status until completed. Once the completion status reaches 100%, the self-calibration process is complete.

To protect the user from performing a self-calibration while the instrument is warming up, a warning message will be issued and calibration will not be performed if self-calibration is attempted before the EX10xxA has been powered on continuously for 60 minutes. This is only a warning, however, and it can be overridden with the **Override Self Calibration Uptime** button that appears when this warning is triggered. In general, this option should not be used, as performing self-calibration prior to complete instrument warm-up could result in degraded measurement performance. However, it would be appropriate to utilize the override in the case where the unit has actually been warmed up, but has simply been subjected to a quick power cycle or reboot.

As noted above, the self-calibration operation places its calculated data in volatile memory, termed "current self cal data." Once present, it can be cleared by utilizing the **Clear Current Self Cal Data** button. This operation clears the volatile data, but does not affect any self cal data that is stored in nonvolatile memory.

Despite having the ability to conduct self-calibration at any time, there may be user applications that require the use of self-calibration, but demand that it create non-volatile data. The **Store Current Self Cal Data As Nonvolatile** button saves the current self cal data to nonvolatile memory, enabling it to be loaded upon instrument power cycle and reset. If this button is pressed when no current self cal data is present, an error is generated. Since the existence of nonvolatile self cal data represents a permanent (although revocable) change from the factory calibration settings, its presence is noted on the **EX10xxA Main** and **Self Calibration** pages.

Previously saved nonvolatile self cal data can be cleared through the use of the **Clear Nonvolatile Self Cal Data** button. This operation does not clear the current self cal data, only that in nonvolatile memory. If this button is pressed when no nonvolatile self cal data is present, an error is generated. Alternatively, previously saved nonvolatile self cal data can be loaded as the current self cal data through the use of the **Load Nonvolatile Self Cal Data** button. If current self cal data previously existed, it is simply overwritten and need not be cleared in advance. If this button is pressed when no nonvolatile self cal data is present, an error is generated.

The ability to conduct a self-calibration is not affected by the existence of current self cal data or nonvolatile self cal data. A performed self-calibration will simply overwrite any current self cal data that already existed. It is not necessary to utilize the **Clear Current Self Cal Data** button prior to performing a new self-calibration. Similarly, the ability to store current self cal data as nonvolatile is not affected by the existence of previously saved nonvolatile data. The old data is simply overwritten and need not be cleared in advance.

NOTE	As part of the self-calibration process, the EX10xxA is reset, returning all configuration parameters to their default values. This reset also clears all volatile self-calibration data.

NOTE	EX1044A/EX1044 system requires hard reboot after self-calibration process so that instrument will be calibrated values gets loaded upon reboot.

NOTE	Self calibration is not applicable for RX10xx since during factory calibration process each instrument will be calibrated in precise manner and calibrated data gets stored in the instrument memory. The instrument has to be sent to factory to perform subsequent calibration after the initial calibration date is expired *Refer below section for* ***Dynamic Calibration in RX1032***

Dynamic Calibration in RX1032

The dynamic calibration should be conducted to calibrate the measurement drift due to the temperature change. For every measurement scan, the values are adjusted based on the system internal temperature.

- The initial calculated Voltage and Temperature coefficients are stored into the RX1032 system.
- At every measurement the measured values are calibrated dynamically using the stored coefficients.
- Based on the present temperature of the system, the values are calibrated and reported.

Get Calibration Files

This action page allows the user to download calibration data directly from the instrument. The “Factory Calibration”, “Self Calibration”, and “Non-Volatile Self Calibration” links become active after a calibration of that type has been performed. Once the link is clicked, an XML file will be downloaded which contains the current calibration data for that type.

Network Configuration

This entry page is used to change the network configuration of the EX10xxA. By default, the EX10xxA will attempt to locate a DHCP server. If one is found, the IP address assigned by the DHCP server will be assumed. Otherwise, after a timeout of 20 seconds, the unit will attempt to obtain an IP address by using Auto IP.

Auto IP is a mechanism for finding an unused IP address in the range 169.254.X.Y where X is in the range 1 through 254 and Y is in the range 0 through 255. The device will first attempt to obtain the specific address 169.254.X.Y, where X and Y are the second-to-last and last octets of the device's MAC address. However, X will be set to 1 if it is 0 in the MAC address, and to 254 if it is 255 in the MAC address. If this address is already in use, the unit will attempt to obtain other IP addresses in a pseudorandom fashion until it finds one that is available.

To illustrate the Auto IP mechanism, Table 5-3 lists the Auto IP default address for some example MAC addresses.

MAC Address	Auto IP Default Address
00:0D:3F:01:00:01	169.254.1.1
00:0D:3F:01:01:01	169.254.1.1
00:0D:3F:01:A3:28	169.254.163.40
00:0D:3F:01:FE:FE	169.254.254.254
00:0D:3F:01:FF:FE	169.254.254.254

TABLE 5-3: AUTO IP DEFAULT ADDRESS ASSIGNMENT

If a static IP address assignment is preferred, one can be optionally assigned via this page. Click on the **Disabled** radio button and then assign a static IP address. Following any changes, a **Hard Reboot** must be conducted to activate them. The network configuration settings are stored in nonvolatile memory and are unaffected by the **Reset Device** or **Hard Reboot** commands.

However, a much more convenient and recommended way to obtain the benefits of a static IP address is to employ DHCP, but assign the instrument a reserved IP address in your company's DHCP server configuration. This reserved address, linked to the EX10xxA's MAC address on the DHCP server, would be assigned to the EX10xxA at power up initialization without having to manually set it on the EX10xxA. The DHCP server configuration provides a centralized, controlled database of assigned IP addresses, preventing accidental assignment of the same IP address to multiple instruments. Consult your company's Information Technology department for assistance.

VXI-11 Device Discovery is also supported by the EX10xxA. This allows all EX1048s on a local network to be found without knowledge of their MAC address or IP address with the use of a broadcast message.

Device Reset button

The device reset button on the rear panel of the EX10xxA can be used to restore default network settings. This is useful for recovery from an incorrect or unknown network configuration. To perform a network reset:

1) Power off the EX10xxA.
2) Press and hold the reset button.
3) Power on the EX10xxA.
4) Continue to hold the reset button for at least 30 seconds.
5) Release the reset button.

The EX10xxA will power up as usual, but will use the default network configuration (DHCP) instead of its previous settings.

Device Identify

This action page provides the instrument's identification information: the IP Address, Hostname, MAC Address, and the status of the Indicator LED. By clicking the **Identify** button, the LAN LED will flash green in order for the user can identify which instrument is being commanded. To return the LED to its normal mode of operation, click on the **Stop Identify** button.

Time Configuration

This entry page is used to change the time configuration of the EX10xxA. As the EX10xxA has no notion of "wall clock" or calendar time and no battery-backed clock or any other mechanism to retain time between reboots and power-cycles, the time must be set by the user. By default, the instrument's time and date at power-up are midnight, January 1, 1999 (as defined by the microcontroller). Three time source options are provided: **PTP** (Precision Time Protocol or IEEE 1588 synchronization), **SNTP** (simple network time protocol), or **manual**. The default time source is PTP.

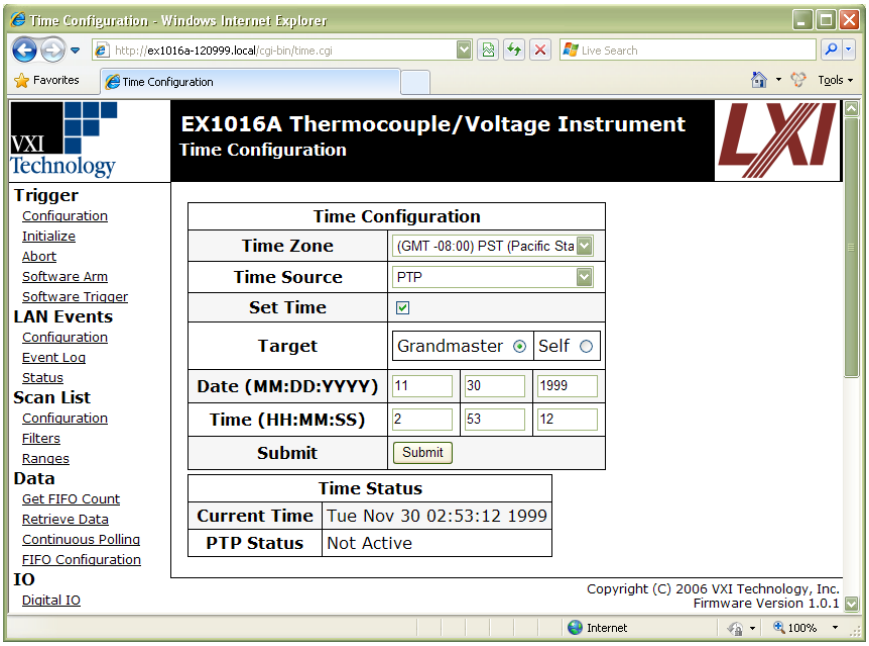


FIGURE 5-6: TIME CONFIGURATION PAGE - PTP

- **Time Zone**: allows the user to select an appropriate time zone.
- **Time Source**: allows the user to set the EX10xxA's date by entering the month (MM), day (DD), and year (YYYY), respectively.
- **Set Time**: for the **PTP** and **Manual Time Source**, allows the user to set the EX10xxA time by entering the hour (HH), minute (MM), and second (SS), respectively. If the EX10xxA is set as an IEEE 1588 Master, the hour should be entered using the 24-hour format.
- **NTP Servers**: for the **SNTP Time Source**, allows the use to add/remove/save NTP servers to which the EX10xxA will synchronize its time.
- **Time Status**: this section of the web page provides the mainframe's current time configuration settings.

Using PTP Time

To configure the EX10xxA to use PTP time as the clock source:

1) Set **Time Source** to **PTP**. The page will change so that it can be configured.
2) Click **Submit** to start using PTP time.

If a PTP master clock does not exist on the network, an LXI device on the network will begin serving as the PTP time master. The **Time Configuration** page allows for PTP time to be set manually. From the **Time Configuration** page, select the **Set Time** checkbox, then select the **Grandmaster** radio button from the **Target** field, and then configure the time by hand. Note that the **Time** field should be completed using a 24-hour format.

To use the EX10xxA as the PTP clock that will serve time on the LXI network, select the **Self** radio button in the **Target** field. The EX10xxA then becomes the PTP master clock. While this ensures measurement synchronicity between the units that to this clock, the overall timestamp accuracy is not as accurate because of the time's hand-set nature. For higher overall timestamp accuracy, use a PTP master clock that is synchronized to a global time source like, such as GPS.

Using SNTP/NTP Time

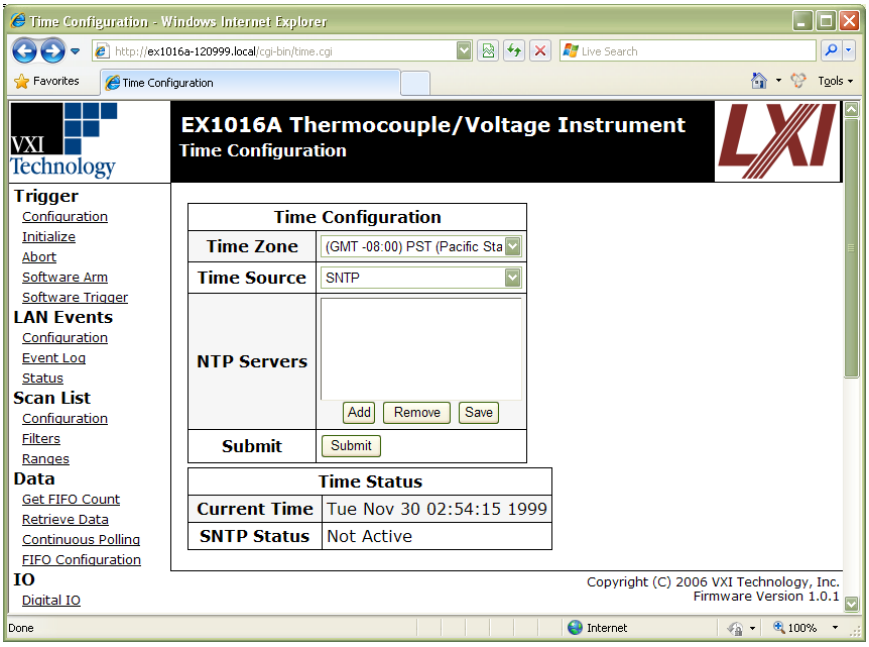


FIGURE 5-7: TIME CONFIGURATION PAGE - SNTP

By selecting **SNTP** from the **Time Source** menu, the EX10xxA can be synchronized with an SNTP server. The **Time Zone** drop menu allows the user to select the local time zone. SNTP, or NTP (Network Time Protocol), servers are specified in the **NTP Servers** field, by IP address or hostname. Once configured, the user must click **Submit** in order for the changes to take effect.

To configure the EX10xxA to use NTP time as its clock source:

1) Select **SNTP** from the **Time Source** field. The page will change so that it can be configured.
2) **Add**/**Remove** NTP servers use the appropriate buttons until the EX10xxA is configured to communicate to the desired server(s).
3) Click the **Save** button to save the server configuration.
4) Click **Submit** to begin using NTP time.

NOTES

1) If the **Submit** button is clicked before saving an NTP configuration, the EX10xxA will start using NTP time, but will use the last previously saved server configuration.
2) Specifying SNTP or NTP servers by hostname requires that the instrument has DNS, either by DHCP or Static IP.

Using the Manual Time Setting

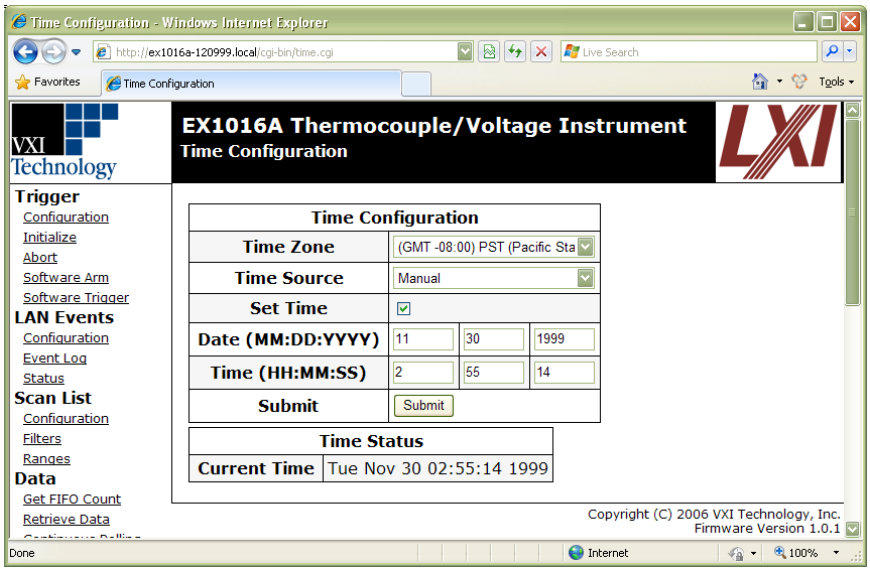


FIGURE 5-8: TIME CONFIGURATION PAGE - MANUAL

If a PTP master or an SNTP server is not available, time can be configured manually by the user by selecting **Manual** in the **Time Source** drop menu. Note, manually set time sources are not very accurate and that this configuration should only be used if timestamp accuracy is not vital to a measurement.

To use a manually set time source:

1) Select **Manual** from the **Time Source** field. The page will change so that it can be configured.
2) Select the **Set Time** checkbox.
3) Enter the required data into the **Date** and **Time** fields that appear. Note that the **Time** field should be entered using the 24-hour format.
4) Click the **Submit** button to begin using the manual time source.

LXI Synchronization

This status page displays the EX10xxAs IEEE 1588 status. The **PTP Grandmaster Clock MAC** field indicates the MAC address of the device on the network to which all other IEEE 1588 compliant devices are synchronized. The **PTP Parent Clock MAC** field identifies the MAC address of the device to which the EX10xxA is synchronized. On a LAN that consists of multiple subnets, this indicates the MAC address of the parent clock on that particular subnet. The **PTP State** field shows the EX10xxA state: master, slave, faulty, disabled, passive, or unlocked.

Two types of wall clock time are available as well. The **Current PTP Time** field indicates the number of seconds since midnight January 1, 1970 UTC (coordinated universal time). The local time can be viewed by observing the **Current Local Time** field.

The **Grandmaster Traceability to UTC** indicates how the grandmaster is synchronized to the UTC. The values provide an indicator of how closely synchronized the grandmaster is to UTC time (see Table 5-4). The Domain field displays the instrument's PTP subdomain. Typically, this indicator is _DFLT. The final field, **Wired-OR Bias,** indicates whether the Wired-OR Bias for each LXI trigger line (LXI0 through LXI7) is enabled or disabled.

Value	Clock Source	UTC Synchronization
ATOM	UTC Atomic Clock	Less than 100 ns
GPS	GPS Receiver	Less than 100 ns
NTP	Network Time Protocol	Less than 50 ms
HAND	Manually/Automatically Set	Less than 10 s
INIT	User Defined	Not specified
DFLT	None	None

TABLE 5-4: UTC GRANDMASTER CLOCK IDENTIFIERS

Please note, the **Current Variance of Parent** field is not currently supported by the EX10xxA.

Reset Device

This action page is used to return all of the EX10xxA's acquisition configuration parameters to their default values. It is most commonly used to return the instrument to a known configuration state prior to the initiation of a new test sequence. Specific affected configuration parameters and their reset values are documented in Table 6-5. With regards to self-calibration data, an instrument reset will clear the current self cal data and then load, if it exists, the nonvolatile self cal data as the current self cal data. For more information on the generation and use of this data, refer to the **Self-calibration** page description in this section.

NOTE An instrument reset clears the FIFO reading memory. All desired acquisition data must be retrieved from the FIFO prior to the issuance of this command.

Hard Reboot

This action page is used to perform a complete instrument reboot, equivalent to that which occurs when the instrument is power cycled. It is most commonly used to accept changes that are made to the network configuration or time configuration settings. In addition, it is suggested that a reboot be performed before conducting a firmware upgrade. Finally, it can be used to perform instrument recovery if the unit is suspected to be in a failed firmware state.

Upgrade Firmware

This action page is used to upgrade the embedded firmware of the EX10xxA. Prior to initiating the firmware upgrade process, a new, uncompressed firmware image must be obtained from VTI Instruments and be accessible from the computer that is connected to the EX10xxA. Unless specifically noted by VTI Instruments, firmware upgrades do not alter the calibration of the EX10xxA.

NOTE Do not power cycle the EX10xxA during the firmware upgrade process. If power is lost during the upgrade, the instrument may be put into an inoperable state, requiring return to the factory. An uninterruptible power supply may be used to avoid this risk.

Perform the following steps to conduct a firmware upgrade:

1) Perform a **Hard Reboot** or a power cycle.
2) Connect to the EX10xxA via the embedded web page.
3) Click on the **Upgrade Firmware** link.
4) Click on the **Proceed** button.
5) Click on the **Browse** button and select the firmware image file to be uploaded to the instrument.
6) Click the **Upload** button to initiate the upgrade process.

NOTE Once the upgrade process is initiated, do not attempt to reconnect to the instrument. Wait until the process is complete, as described below, before reconnecting.

The process takes approximately 5 minutes to complete. Once the instrument has rebooted, reconnect and confirm on the **EX10xxA Main** page that the firmware revision level has been properly updated.

LOCKING MENU

The Locking menu is used to acquire, release, and break a lock on the EX10xxA's acquisition system.

Lock

This action page attempts to acquire a lock. It is good practice to lock the instrument whenever protected operation is desired, but it is mandatory in order to perform self-calibration. If the EX10xxA has been previously locked by another host, the lock will fail, and an error message will be returned.

Unlock

This action page attempts to release a previously acquired lock. It will only release a lock that was acquired by the same host. It will not release a lock acquired by a different host. In that case, the unlock will fail, and an error message will be returned. An error message is also returned if the unlock command is issued when the instrument is not in the locked state.

Break Lock

This action page breaks the lock on the EX10xxA, regardless of the host that originally acquired it. It does not automatically acquire a lock in the same operation. If desired, that must be done with a separate **Lock** command.

ADVANCED MENU

The Advanced menu is used to configure the EX10xxA to employ user-defined thermocouple polynomial equations or utilize user-defined CJC temperatures. Neither of these functions is required for normal operation with standard thermocouples.

User Conversions

This entry page is used to enter user-defined thermocouple polynomial coefficients. Up to two unique sets of coefficients can be entered. For more details, see *User-defined Conversions* in Section 3. The entry of user-defined coefficients does not automatically enable their use. The enabling is done by setting the EU conversion to User0 or User1 in the **Scan List…Configuration** page.

User CJC Temp

This entry page is used to enter and enable user-defined CJC temperatures. Each channel can be associated with a unique value and be independently enabled with regards to its use. If enabled, the user-defined CJC temperature will be used in the thermocouple calculations instead of the internally measured one. The entry of external CJC values and their enabling are disjoint functions. That is, the entry of a value does not automatically enable its use, and the disabling of a previously enabled channel does not clear the value. All entries must be in units of ºC.

NOTE	This feature must be used with care, as the input channel data contains no indication that it was calculated with an external CJC value instead of an internal one.

CONFIGURATION EXAMPLES

Example #1

This example demonstrates a typical setup sequence for a static monitoring application with an alarm. The requirements of the application are:

- Input connections are five type J thermocouples connected to channels 0-4 and five type K thermocouples connected to channels 5-9.
- The temperatures to be monitored are slow-moving, such that the 4 Hz filter frequency is suitable.
- Measurement data is required to be in units of ºF.
- CJC temperature reporting is desired.
- Data is to be acquired at a rate of 20 readings/second whenever DIO Channel 0 is high.
- Once initiated, the test should continue until aborted.
- DIO Channel 1 is to be used as an alarm indication, in which it needs to transition high and stay high if any of the 10 input channels exceeds 100 ºF.

The exact steps to be performed to satisfy this application are:

Configure the Scan List

The scan list is configured with the **Scan List…Configuration** page. Click the **Disabled** link to clear all previously enabled channels and then check the checkboxes of channels 0 through 9. With the appropriate radio button, set the EU conversion of channels 0 through 4 to **J** and channels 5 through 9 to **K**. Click the **Apply** button to accept the changes.

Configure the Filter Frequencies

The filters are configured with the **Scan List…Filters** page. Click the 4 Hz radio button for channels 0 through 9. Alternatively, since all enabled channels require the same filter setting, the **4 Hz** link can be used to simultaneously set all channels. Click the **Apply** button to accept the changes.

Configure the FIFO

The FIFO is configured with the **Data…FIFO Configuration** page. Select the **Temperature Units** selection of **F**. Check the **Report CJC Temperatures** checkbox. Click the **Apply** button to accept the changes.

Configure the Trigger Model

The trigger model is configured with the **Trigger…Configuration** page. Click the **Default** button. Expand the Arm branch to expose the Arm source. Deselect any currently checked sources. Expand the Digital IO branch and select **Pos. Level** of DIO Channel 0. Within the Trig source, select **Timer**. Enter a **Timer Interval** of **0.05**. Check the **Init Continuous** checkbox. Click the **Apply** button to accept the changes.

Configure the Limit System

Limit conditions are configured with the **Limits…Limits** page. Using the **Set** pull-down selection, select Set: **0**. For channels 0-9, check the **Manual** checkbox and enter an upper limit value of **100**. Click the **Apply** button to accept the changes. These limit conditions are linked to the DIO channel through the **Limits…DIO Limit Events** page. Using the **DIO Channel** pull-down selection, select **DIO Channel: 1**. Check the **Set 0 Upper** checkboxes for Input Channels 0-9. Check the **Latch** checkbox. Click the **Apply** button to accept the changes.

Initiate the Trigger Model

The acquisition sequence is initiated with the **Trigger…Initialize** page.

Example #2

This example demonstrates a typical setup sequence for a transient monitoring application with an alarm. The requirements of the application are:

- Input connections are 48 type E thermocouples connected to channels 0 through 47.
- The temperatures to be monitored are transient in nature, such that the 1 kHz filter frequency is suitable.
- Measurement data is required to be in units of ºC.
- Timestamp reporting is desired, but CJC temperature reporting is not.
- Data is to be acquired at a rate of 1000 readings/second for 10 seconds, triggered by a positive edge transition on channel 0 of the trigger bus.
- Once the 10 seconds of data is collected, the test is complete.
- DIO Channel 7 is to be used as an alarm indication, in which it needs to transition low if any of the 48 channels exceeds 50 ºC.

The exact steps to be performed to satisfy this application are:

Configure the Scan List

The scan list is configured with the **Scan List…Configuration** page. Click the **Enabled** link to enable all 48 channels. Since all channels require the same EU conversion, click the **E** link. Click the **Apply** button to accept the changes.

Configure the Filter Frequencies

The filters are configured with the **Scan List…Filters** page. Since all channels require the same filter setting, click the **1 kHz** link. Click the **Apply** button to accept the changes.

Configure the FIFO

The FIFO is configured with the **Data…FIFO Configuration** page. Select the **Temperature Units** selection of **C**. Uncheck the **Report CJC Temperatures** checkbox. Check the **Report Timestamps** checkbox. Click the **Apply** button to accept the changes.

Configure the Trigger Model

The trigger model is configured with the **Trigger…Configuration** page. Click the **Default** button. Expand the Arm branch to expose the Arm source. Deselect any currently checked sources. Expand the VTI Trig. Bus branch and select **Pos. Edge** of VTB Channel 0. Within the Trig source, select **Timer**. Set a **TRIG Count** of **10000**. Enter a **Timer Interval** of **0.001**. Uncheck the **Init Continuous** checkbox. Click the **Apply** button to accept the changes.

Configure the Limit System

Limit conditions are configured with the **Limits…Limits** page. Using the **Set** pull-down selection, select **Set**: **0**. For channels 0-47, check the **Manual** checkbox and enter an upper limit value of 50. Click the **Apply** button to accept the changes. These limit conditions are linked to the DIO channel through the **Limits…DIO Limit Events** page. Using the **DIO Channel** pull-down selection, select channel **7**. Since all channels are required in the limit evaluation, click the **Set 0 Upper All** link. Check the **Invert** checkbox. Uncheck the **Latch** checkbox. Click the **Apply** button to accept the changes.

Initiate the Trigger Model

The acquisition sequence is initiated with the **Trigger…Initialize** page.

SECTION 6

PROGRAMMING

PROGRAMMING SEQUENCE

The EX10xxA instrument driver is a VXI *plug&play* driver that is composed of top-level C functions. The Figure 6-9 describes the typical steps performed in programming an acquisition sequence on the EX10xxA. While the flow chart implies an order dependence to the functions, none truly exists. Each function operates in a disjoint manner. The flow chart, however, does present a logical flow of operations. For many applications, some of the detailed steps would not be required, either because the default parameters are sufficient or the associated functionality is simply not required.

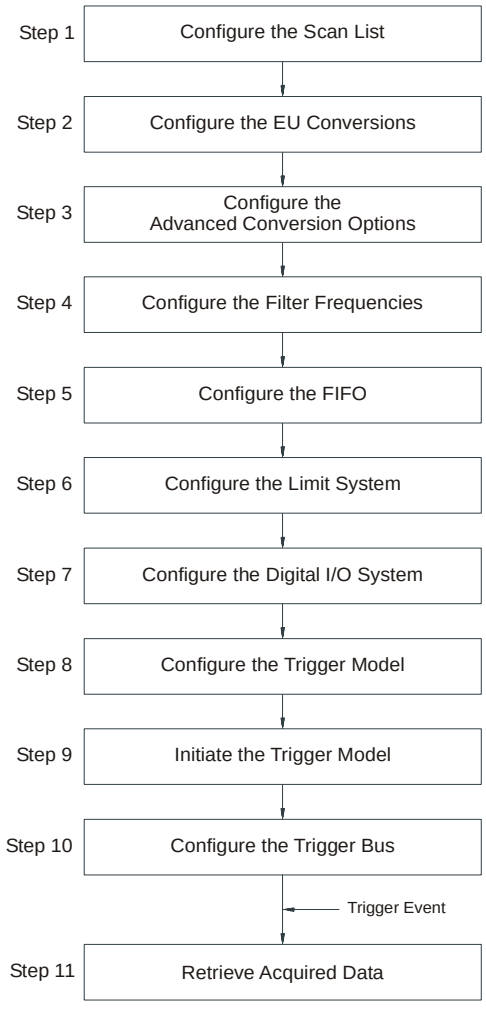


FIGURE 6-9: PROGRAMMING SEQUENCE DIAGRAM

DEFAULT SETTINGS

The default instrument settings after an instrument reset or a power cycle are listed in Table 6-5. Many programming applications do not require parameter changes from the default settings and can be made far simpler by the elimination of redundant commands. The EX10xxA can be returned to the reset state at any time by calling the vtex10xxA_reset function.

TRIGGER CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Init Continuous	Disabled
Timer Interval (seconds)	0.1
Arm Source	Immediate
Arm Count	1
Arm Delay (seconds)	0
Trig Source	Timer
Trig Count	1
Trig Delay (seconds)	0
SCAN LIST CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Scan List (enabled channels)	0
EU Conversions	mV (voltage)
Filter (hertz)	4.0
FIFO CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
CJC Temperature Reporting	Disabled
Timestamp Reporting	Disabled
Temperature Units	°C
Blocking Mode	Disabled
DIGITAL I/O CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Output State	0
Output Enable	Disabled
TRIGGER BUS CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Output State	0
Output Enable	Disabled
LIMITS CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Lower Limit of Limit Set 0	-0.066
Upper Limit of Limit Set 0	0.066
Limit Set 0, Manual	Disabled
Lower Limit of Limit Set 1	-1e+38
Upper Limit of Limit Set 1	1e+38
DIO Limit Events	Disabled
DIO Limit Events, Invert	Disabled
DIO Limit Events, Latch	Disabled
DEVICE CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
Current Self Cal Data	Cleared
Network Configuration	Unaffected
Time Configuration	Unaffected
Lock	Unaffected

TABLE 6-5: DEFAULT SETTINGS

ADVANCED CONFIGURATION RESET VALUES	
CONFIGURATION PARAMETER	**RESET VALUE**
User-defined Conversion Coefficients	0
User-defined CJC Temperatures (values)	0
User-defined CJC Temperatures (enable)	Disabled

TABLE 6-1 (CONTINUED): DEFAULT SETTINGS

EX10XXA BACKWARD COMPATIBILITY

If an EX10xxA will replace an EX1048 in a test station, the existing program can be easily modified to work seamlessly with the EX10xxA. By replacing all references of "vtex1048" with "vtex10xxA" and using the EX1048 rev 2 *plug&play* driver , the EX10xxA can utilize the existing code. Likewise, user applications developed for the EX10xxA can function with the EX10xxA hardware by loading existing EX10xxA firmware.

While EX10xxA *plug&play* driver is functionally compatible with the EX1048 driver, some notable programming differences arise due to hardware/firmware difference. For example, if an application utilizes the high-speed streaming interface, minor application modification are needed in order to get the correct time stamps from EX10xxA products.

- EX10xxA *plug&play* driver has "vtex10xxA" prefix, while EX1048 has "vtex1048" prefix.
- The time stamp data obtained from high-speed, streaming interface shall have a unit of nanoseconds (ns) with EX10xxA products while they are microseconds (µs) with EX1048. The nanosecond time stamp is required by LXI Class A specification. Alternatively, the user can use new vtex10xxA_enable_streaming_dataEx function with **legacy_mode** = TRUE to obtain the microseconds time stamp. When the user uses the EX1048 rev 2 *plug&play* driver with EX1048A, the time stamp will be returned in microseconds.
- The ID string returned by will be "EX1048A", "EX1032A","EX1016A" or "EX1000A" instead of "EX1048".
- EX10xxA IVI-COM driver will have "VTEX10xxA" as an instrument specific interface prefix, while EX1048 IVI-COM driver has "VxiTechEX1048".
- EX10xxA IVI-C driver API has "vtiex10xxA" as a function prefix while EX1048 IVI-C driver has "vtiex1048" as function prefix.

The EX10xxA driver, however, does not support the EX1048.

GENERAL COMMANDS / QUERIES

These commands are used for general instrument setup and communication. Details on full command syntax are provided in Section 5.

Initializing the device

The **vtex10xxA_init** function is used to establish communication with the EX10xxA and must be performed before any other command. The key parameter of this function is the IP address of the EX10xxA to which to connect. Also specified within this function are options to confirm the identity of the connected instrument and to reset the instrument upon connecting. The output of this function is the session handle assigned to the connection, which is a unique identifier necessary in all subsequent communications. Consequently, sessions to multiple instruments can be opened within the same application, each uniquely identified by their session handle.

The **vtex10xxA_revision_query** function is used to obtain the release revisions of the instrument driver and the embedded firmware of the connected EX10xxA.

The **vtex10xxA_close** function is used to terminate an established communication link with the EX10xxA and should be performed at the conclusion of the test application. Part of its execution is the unlocking of the instrument, leaving it in the proper state for the next application.

Resetting the device

The **vtex10xxA_reset** function is used to return all of the EX10xxA's acquisition configuration parameters to their default values. It is most commonly used to return the instrument to a known configuration state prior to the initiation of a new test sequence. Specific affected configuration parameters and their reset values are documented in Table 6-5. With regards to self-calibration data, an instrument reset will clear the current self cal data and then load, if it exists, the nonvolatile self cal data as the current self cal data. For more information on the generation and use of this data, refer to the **Performing Self-Calibration** description in this section.

NOTE	An instrument reset clears the FIFO reading memory. All desired acquisition data must be retrieved from the FIFO prior to the issuance of this function.

Performing Self-Calibration

As discussed in *Self-calibration* in Section 3, the measurement performance of the EX10xxA is significantly improved through the periodic use of self-calibration. The self-calibration process completes quickly and does not require removal of the actual input connections or the use of external equipment, making it convenient to run often. In order to take full advantage of the self-calibration system, it is important to understand how the self-calibration constants are generated, used, and stored.

Self-calibration does not overwrite, modify, or take the place of the instrument's nonvolatile calibration constants generated by a full calibration. Instead, it generates an additional set of calibration constants that are applied to the measurement calculation after the full calibration constants. When self-calibration is performed, it places its calculated data in volatile memory. This data is termed the "current self cal data" and is used automatically by the instrument. No additional function is necessary. Because it is in volatile memory, however, it will be cleared upon an instrument reset or power cycle. There is the ability, however, to store the current self cal data into nonvolatile memory.

In order to perform a self-calibration, a lock on the instrument must first be acquired. For more information on locking, refer to the **Acquiring a Lock** description in this section. If self-calibration is attempted without the acquisition of a lock, an error will be generated. The self-calibration process is initiated with the **vtex10xxA_self_cal_init** function. Once started, its percentage completion status is accessible through the **vtex10xxA_self_cal_get_status** query. Other than this status check, additional instrument driver calls should not be performed during self-calibration. Once the completion status reaches 100%, the self-calibration process is complete.

To protect the user from performing a self-calibration while the instrument is warming up, an error will be generated and calibration will not be performed if self-calibration is attempted before the EX10xxA has been powered on continuously for 60 minutes. This is only a warning, however, and it can be overridden. In general, this option should not be used, as performing self-calibration prior to complete instrument warm-up could result in degraded measurement performance. However, it would be appropriate to utilize the override in the case where the unit has actually been warmed up, but has simply been subjected to a quick power cycle or reboot. Details on the override mechanism are provided in the **vtex10xxA_self_cal_init** function description in Section 5.

NOTE	The self-calibration uptime requirement is in place to protect the measurement integrity of the instrument and should only be overridden with caution. In order to ensure that the override is intentional, it is strongly recommended that user intervention be required in the software application to employ it.

As noted above, the self-calibration operation places its calculated data in volatile memory, termed "current self cal data." Once present, it can be cleared by utilizing the **vtex10xxA_self_cal_clear** function. This operation clears the volatile data, but does not affect any self cal data that is stored in nonvolatile memory.

Despite having the ability to conduct self-calibration at any time, there may be user applications that require the use of self-calibration, but demand that it create nonvolatile data. The **vtex10xxA_self_cal_store** function saves the current self cal data to nonvolatile memory, enabling it to be loaded upon instrument power cycle and reset. If this function is sent when no current self cal data is present, an error is generated. Since the existence of nonvolatile self cal data represents a permanent (although revocable) change from the factory calibration settings, its presence is able to be ascertained with the **vtex10xxA_self_cal_is_stored** query.

Previously saved nonvolatile self cal data can be cleared through the use of the **vtex10xxA_self_cal_clear_stored** function. This operation does not clear the current self cal data, only that in nonvolatile memory. If this function is sent when no nonvolatile self cal data is present, an error is generated. Alternatively, previously saved nonvolatile self cal data can be loaded as the current self cal data through the use of the **vtex10xxA_self_cal_load** function. If current self cal data previously existed, it is simply overwritten and need not be cleared in advance. If this function is sent when no nonvolatile self cal data is present, an error is generated.

The ability to conduct a self-calibration is not affected by the existence of current self cal data or nonvolatile self cal data. A performed self-calibration will simply overwrite any current self cal data that already existed. It is not necessary to utilize the **vtex10xxA_self_cal_clear** function prior to performing a new self-calibration. Similarly, the ability to store current self cal data as nonvolatile is not affected by the existence of previously saved nonvolatile data. The old data is simply overwritten and need not be cleared in advance.

NOTE As part of the self-calibration process, the EX10xxA is reset, returning all configuration parameters to their default values.

NOTE	Self calibration is not applicable for RX10xx since during factory calibration process each instrument will be calibrated in precise manner and calibrated data gets stored in the instrument memory. The instrument has to be sent to factory to perform subsequent calibration after the initial calibration date is expired. *Refer below section for **RX1032 Dynamic Calibration***

NOTE	EX1044A/EX1044 system requires hard reboot after self-calibration process so that instrument will be calibrated values gets loaded upon reboot.

Dynamic Calibration in RX1032

The dynamic calibration should be conducted to calibrate the measurement drift due to the temperature change. For every measurement scan, the values are adjusted based on the system internal temperature.

- The initial calculated Voltage and Temperature coefficients are stored into the RX1032 system.
- At every measurement the measured values are calibrated dynamically using the stored coefficients.
- Based on the present temperature of the system, the values are calibrated and reported.

Acquiring a Lock

By default, the EX10xxA allows unrestricted operation from multiple hosts. While this offers a high level of user flexibility, there are instances where protected operation is desirable, if not required. For these applications, the EX10xxA can be "locked," meaning that it will accept functions from only the host IP address that issued the **vtex10xxA_lock** function. With the EX10xxA in this mode, other host connections that attempt functions will be denied. Self-calibration, in fact, requires the acquisition of a lock prior to its initiation.

Once a lock has been acquired, it is released with the **vtex10xxA_unlock** function. The lock status is not affected by the **vtex10xxA_reset** function, and it cannot be used to release a lock.

By design, the locking mechanism is able to be overridden by a secondary host that issues a **vtex10xxA_break_lock** function. Thus, the lock provides a warning to other users that the unit is in a protected operation state, but not absolute security. This allows for instrument recovery if the locking IP address would become disabled. Breaking a lock, however, does not automatically acquire it. That must be done with a separate **vtex10xxA_lock** function.

The lock status of the instrument is queried with the **vtex10xxA_check_lock** query. The response to the query not only indicates whether the unit is locked, but also if the lock is owned by the issuer of the query.

CONFIGURE THE SCAN LIST

The scan list is defined with the **vtex10xxA_set_scanlist** function. The EX10xxA can be configured to include from 1 to all 48 of its input channels in the scan list. A valid scan list consists of:

- at least one channel
- no more than 48 channels
- no repeated channels

The channels in the scan list can be listed in any order, but will be scanned in numerical order when the scan list is executed.

Example #1: This code block configures a five-channel scan of channels 0 through 4 in sequential order.

```
ViInt32 channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_scanlist(vi, channels, 5);
```

Example #2: This code block configures a five-channel scan of channels 0 through 4 in reverse order.

```
ViInt32 channels[5] = { 4, 3, 2, 1, 0 };
vtex10xxA_set_scanlist(vi, channels, 5);
```

The current scan list is queried with the Vtex10xxA_get_scanlist function.

CONFIGURE THE EU CONVERSIONS

The EU conversion for each thermocouple channel is configured with the **vtex10xxA_set_channel_conversion** function. A channel's EU conversion can be configured regardless of its inclusion in the scan list, and multiple channels can be assigned to a specific conversion type within one function. However, each unique conversion type must be set with a separate function.

Example: This code block configures channels 0 through 4 for thermocouple type E and channels 5 through 8 for thermocouple type T.

```
#define TYPE_E    0x04
#define TYPE_T    0x03
ViInt32 e_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_channel_conversion(vi, e_channels, 5, TYPE_E);
ViInt32 t_channels[4] = { 5, 6, 7, 8 };
vtex10xxA_set_channel_conversion(vi, t_channels, 4, TYPE_T);
```

The current EU conversion assignment for any channel is queried with the **vtex10xxA_get_channel_conversion** query.

CONFIGURE THE RANGE

Use the **vtex10xxA_set_channel_range** function to set the range for EX10xxA voltage input channel on a per channel basis. When a channel is set to perform voltage measurements, the input connector determines the range available. For thermocouple channels, only the 67 mV range can be used. Any other range will generate an error. Voltage channels can be set to any voltage range, with 10 V being the default value.

```
//sets an array of the 1032's voltage channels to 1.0 volt range
        ViInt32 ex1032_volt_channels [16] = { 0, 1, 2, 3, 4 ,5 ,6 ,7 ,8 ,9 ,10 ,11 ,12 ,13 ,14 ,15};
        vtex10xxA_set_channel_ranges(vi, ex1032_volt_channels, 16, 1.0);
```

CONFIGURE THE ADVANCED CONVERSION OPTIONS

To facilitate its use, the EX10xxA provides a high level of hardware and software integration. A high performance internal cold junction mechanism allows for direct connection of thermocouple wire, and embedded polynomial coefficients for all standard thermocouple types provides compensated temperature readings without external mathematical manipulation. It does, however, also support the advanced configuration options of employing an external cold junction and employing custom thermocouple conversions.

Employing an external cold junction

The EX10xxA accommodates the use of external cold junctions that are maintained and measured by the user. In this application, the cold junction temperature in ºC is entered into the EX10xxA and enabled on a per channel basis. That is, the use of internal and user-defined CJC inputs can be mixed throughout the unit. User-defined CJC temperatures are entered with the **vtex10xxA_set_user_cjc_temp** function. A channel's CJC temperature can be configured regardless of its inclusion in the scan list, and multiple channels can be assigned to the same temperature within one function. However, each unique temperature must be set with a separate function.

The use of a user-defined CJC temperature is enabled on a per channel basis with the **vtex10xxA_set_user_cjc_enable** function. If enabled, the user-defined CJC temperature will be used in the thermocouple calculations instead of the internally measured one. The entry of external CJC values and their enabling are disjoint functions. That is, the entry of a value does not automatically enable its use, and the disabling of a previously enabled channel does not clear the value.

NOTE	These functions should only be used when the thermocouple cold junction is made external to the EX10xxA. This feature must be used with care, as the input channel data contains no indication that it was calculated with an external CJC value instead of an internal one.

The current user-defined CJC temperature and enable status for any channel are queried with the **vtex10xxA_get_user_cjc_temp** and **vtex10xxA_get_user_cjc_enable** queries, respectively.

Employing custom thermocouple conversions

The EX10xxA nominally accepts standard thermocouple types and performs its thermocouple calculations using polynomial coefficients from the NIST ITS-90 Thermocouple Database. In some applications, however, a user may want to override the embedded coefficients with a user-defined coefficient set. One reason to do this is if the transfer function of the specific thermocouple being used has been completely characterized to an accuracy level that exceeds standard thermocouple limits of error. Another reason to do this is if a non-standard thermocouple is used. Up to two unique sets of coefficients can be entered. Specifically, the use of custom thermocouple equations requires the user to know or generate the coefficients for two conversion polynomials.

The *forward conversion polynomial* is used to convert a CJC temperature into a compensating cold junction voltage and has the form of:

$$E = c_0 + c_1 * t^1 + c_2 * t^2 + \ldots + c_{12} * t^{12}$$

where E is in volts, t is in ºC, and c_0 - c_{12} are the coefficients.

The *inverse conversion polynomial* is used to convert a compensated input voltage into temperature and has the form of:

$$t = d_0 + d_1 * E^1 + d_2 * E^2 + \ldots + d_{12} * E^{12}$$

where E is in volts, t is in ºC, and d_0 - d_{12} are the coefficients.

The coefficient sets are entered with the **vtex10xxA_set_user_conversion** function. This function accepts the following parameters:

- an integer value of 9 or 10, corresponding to the EU conversion type (User0 or User1, respectively) to which the entered coefficients are assigned
- an array of forward conversion polynomial coefficients
- an integer value corresponding to the length of the forward coefficients array
- an array of inverse conversion polynomial coefficients
- an integer value corresponding to the length of the inverse coefficients array

Coefficients that are not specifically defined are automatically set to 0. The current values for each coefficient set are queried with the **vtex10xxA_get_user_conversion** query.

NOTE	The entry of user-defined coefficients does not automatically enable their use. The enabling is done by setting the EU conversion to User0 or User1.

CONFIGURE THE VOLTAGE MEASUREMENT CHANNELS

The EX10xxA voltage channel can be configured to perform either general purpose voltage measurement or thermocouple measurement. When they are used for the thermocouple measurement, the user can use the same EU conversion functions. However, the user must use either the user CJC temperature functionality or must connect an external thermistor to the voltage connector (see *Voltage Connections* for more information) when using temperature EUs.

To use these channels for the general purpose voltage measurement, the user must set the EU conversion to VTEX10XXA_CONV_MV using **vtex10xxA_set_channel_conversion** function. The measurement data is returned in units of volts by default, however, custom engineering units can be created using a linear transform specified by the **vtex10xxA_set_linear_correction** function.

CONFIGURE THE FILTER FREQUENCIES

The hardware filter for each channel is configured with the **vtex10xxA_set_filt_freq** function. A channel's hardware filter can be configured regardless of its inclusion in the scan list, and multiple channels can be assigned to a specific filter frequency within one function. However, each unique filter frequency must be set with a separate function.

Example: This code block configures channels 0 through 4 for a 4 Hz filter and channels 5 through 8 for a 500 Hz filter.

```
ViInt32 low_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_filt_freq(vi, low_channels, 5, 4.0);
ViInt32 high_channels[4] = { 5, 6, 7, 8 };
vtex10xxA_set_filt_freq(vi, high_channels, 4, 500.0);
```

The current hardware filter assignment for any channel is queried with the **vtex10xxA_get_filt_freq** query.

CONFIGURE THE FIFO

The **vtex10xxA_set_fifo_config** function is used to set the data format and overflow behavior of the FIFO memory. By default, the data returned during data retrieval is limited to the input channel readings and the absolute time of scan initiation. While measured with every scan, the CJC temperature values are not returned unless enabled to do so. Reported CJC temperatures are always in units of ºC, regardless of the units of the channel readings. In addition, the returned data can be enabled to include delta timestamps per channel. Specifically, the timestamp represents the time, in increments of 100 ns, between the specific channel measurement and the scan initiation time.

By default, the channel readings are returned with units of ºC. Optionally, the units can be set to ºF. This setting has no effect on the reported CJC data or data for input channels that are configured for an EU conversion of mV.

In addition to data formatting, this function is used to specify the system behavior if the reading buffer fills to its maximum capacity during scanning. With blocking mode enabled, additional readings are discarded, leaving the contents of the buffer intact. With blocking mode disabled, the buffer becomes circular, in that additional readings overwrite the oldest readings.

The current FIFO configuration is queried with the **vtex10xxA_get_fifo_config** function.

NOTE	Regardless of their enabling, the CJC temperature and delta timestamp information is not accessible through the **vtex10xxA_read_fifo** function. They are available through the streaming interface, and the parameter settings apply for it.

CONFIGURE THE LIMIT SYSTEM

The EX10xxA features two sets of programmable limits. These limits, termed limit set 0 and limit set 1, are programmable on a per channel basis. Manipulation of the limit set values is slightly different for each limit set, as limit set 0 offers enhanced functionality.

By default, the values in limit set 0 are set automatically, based on the EU conversion and units selection for each channel. Specifically, the upper and lower limit values are set to the upper and lower values of the EX10xxA measurement range, as specified in Table 3-2. For example, if input channel 0 is configured for an E type conversion and a units selection of ºC, its upper and lower limit values will be +900 and -200, respectively. To override the automatic setting and set the values manually on a per channel basis, manual control must first be enabled with the **vtex10xxA_set_limit_set0_manual** function. A channel's manual entry control can be set regardless of its inclusion in the scan list, and multiple channels can be configured within one function. However, each unique control value must be set with a separate function.

Once enabled for manual entry, a channel's limit set 0 values are set with the **vtex10xxA_set_limit_set0** function. A channel's limit values can be set regardless of its inclusion in the scan list, and multiple channels can be assigned to the same limit values within one function. However, each unique combination of limit values must be set with a separate function.

Example: This code block configures channels 0 through 4 for manual limit values of 0 and 100.

```
ViInt32 e_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_limit_set0_manual(vi, e_channels, 5, 1);
vtex10xxA_set_limit_set0(vi, e_channels, 5, 0, 100);
```

Conversely, disabling manual limit control on a channel that had been enabled will automatically set the limit set 0 values for that channel, based on its current EU conversion and units selection.

The current manual entry control and limit set 0 values for any channel are queried with the **vtex10xxA_get_limit_set0_manual** and **vtex10xxA_get_limit_set0** queries, respectively.

In contrast, limit set 1 operates in manual mode only. A channel's limit set 1 values are set with the **vtex10xxA_set_limit_set1** function. A channel's limit values can be set regardless of its inclusion in the scan list, and multiple channels can be assigned to the same limit values within one function. However, each unique combination of limit values must be set with a separate function.

The current limit set 1 values for any channel are queried with the **vtex10xxA_get_limit_set1** query.

NOTE	Limit sets operating in manual mode do not automatically adjust for changes in EU conversion or units selection. For proper operation, limit values should be set after EU conversion and units selections are made and reset upon subsequent configuration changes.

Once an acquisition has been completed, the **vtex10xxA_get_accum_limit_status** query is used to obtain the accumulated limit status of all 48 channels. The response to this query is four arrays of 48 Boolean values each. The four arrays correspond to the set 0 lower limit, set 0 upper limit, set 1 lower limit, and set 1 upper limit, respectively. The returned values represent, on a per channel basis, any excursion of the measurement data over their respective limits since the last trigger initialize. Specifically, a returned "1" indicates that the limit was exceeded, while a returned "0" indicates that it was not. As implied, limit status is cleared as part of the **vtex10xxA_init_imm** function. Limit status is returned for all channels, regardless of their inclusion in the scan list. The limit status for unscanned channels is always 0.

CONFIGURE THE DIGITAL I/O SYSTEM

The digital I/O port on the EX10xxA can be used for input and output operations. Each DIO channel can be independently programmed with regards to its output functionality, its static level to assume when enabled as an output, and pulse operation.

The **vtex10xxA_get_dio_input** query is used to obtain the measured digital level (0 or 1) of the DIO channels. The input monitoring functionality operates regardless of the simultaneous use of the channels as outputs. The response to this query is a decimal value from 0 to 255 that represents the 8-bit value of the port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.

Example: This code block queries the state of the I/O port and reports the level of bits 7, 4, and 0.

```
ViInt32 dio_in;
vtex10xxA_get_dio_input(vi, &dio_in);
if (dio_in & 0x80)
    printf("Bit 7 is high");
else printf("Bit 7 is low");
if (dio_in & 0x10)
    printf("Bit 4 is high");
else printf("Bit 4 is low");
if (dio_in & 0x01)
    printf("Bit 0 is high");
else printf("Bit 0 is low");
```

The **vtex10xxA_set_dio_output** function is used to set the static level that each channel of the digital I/O port will assume if/when enabled as an output. Enabling, however, is done with the **vtex10xxA_set_dio_output_enable** function (see below). This function accepts a value that represents the desired state of the 8-bit port, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.

The **vtex10xxA_set_dio_output_enable** function enables or disables the output functionality of each channel of the digital I/O port. Once enabled, a channel will assume the level specified with the **vtex10xxA_set_dio_output** function (see above). When not enabled as an output, a channel becomes tri-stated. Input functionality on each channel is constant regardless of its output functionality. This function accepts a value that represents the desired output enable state of the 8-bit port, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.

Example: This code block sets bit 7 (high) and bit 6 (low) of the I/O port and then enables them as outputs.

```
vtex10xxA_set_dio_output(vi, 0x80);
vtex10xxA_set_dio_output_enable(vi, 0xC0);
```

The **vtex10xxA_get_dio_output** query is used to obtain the programmed output state of the DIO channels. The response to this query is a decimal value from 0 to 255 that represents the programmed output state of the 8-bit port. However, since the outputs must be enabled, it does not necessarily represent the true output state. The **vtex10xxA_get_dio_output_enable** query is used to obtain the output enable state of each channel. The response to this query is a decimal value from 0 to 255 that represents the output enable state of the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.

Example: This code block queries the output state and output enable state of the I/O port and reports the states of bit 4.

```
ViInt32 dio_out;
ViInt32 dio_outen;
vtex10xxA_get_dio_output(vi, &dio_out);
vtex10xxA_get_dio_output_enable(vi, &dio_outen);
if (dio_out & 0x10)
    printf("Bit 4 is set high");
else printf("Bit 4 is set low");
if (dio_outen & 0x10)
    printf("Bit 4 is enabled");
else printf("Bit 4 is not enabled");
```

The **vtex10xxA_set_dio_pulse** function is used to generate a 1 μs pulse on selected channels of the digital I/O port. The pulse will occur only if the selected channels are enabled as outputs. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate function. This function accepts a value that represents the channels to be pulsed within the 8-bit port, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to DIO channel 7, and the LSB corresponds to DIO channel 0.

Example: This code block sets bit 7 to a static level of low and then generates a high-going pulse.

```
vtex10xxA_set_dio_output(vi, 0x00);
vtex10xxA_set_dio_output_enable(vi, 0x80);
vtex10xxA_set_dio_pulse(vi, 0x80);
```

Configure DIO Limit Events

In addition to performing simple input and output operations, the digital I/O port can also be linked to reflect the state of input channel limit evaluations. This linkage is termed a DIO limit event. DIO limit events are programmable on a per channel basis; that is, each of the 8 DIO channels can be configured with a unique set of operational characteristics. In nominal operation, a DIO channel that is linked to an input channel's limit evaluation will transition from low to high whenever the limit is exceeded.

The linkage of a DIO channel to a specific limit condition on a specific input channel is accomplished with the vtex10xxA_set_dio_limit_event function. Multiple linkages per DIO channel are allowed and are logically OR'ed together. That is, a DIO channel that is linked to four input channel limit evaluations will transition whenever any of the four limits are exceeded. Multiple linkages can be created on the same input channel and/or spanning multiple input channels.

This function accepts the following parameters:

- an integer value representing the DIO channel number (0 through 7)
- an array of 48 4-bit integer values representing, on a per input channel basis, the linking of limit evaluations to any of the 4 limit conditions. Within the 4-bit field, the order of the values is: limit set 0 lower, limit set 0 upper, limit set 1 lower, limit set 1 upper. The values can be specified in either decimal (0 – 15) or hex (0x00 – 0x0F). Channel 0 through channel 47 are represented in array elements [0] through [47], respectively.

Example: This code block links DIO channel 7 to set 1 lower limits on input channels 12 and 14 and set 0 upper limit on input channel 1

```
ViUInt16 limit_masks[48] =
{0,4,0,0,0,0,0,0,0,0,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
vtex10xxA_set_dio_limit_event(vi, 7, limit_masks);
```

The DIO limit event and the scan list configuration controls are autonomous. That is, specifying a linkage to a specific input channel does not automatically enable it in the scan list. Similarly, removing an input channel from the scan list does not automatically remove it as a linked limit condition. DIO limit event linkages to unscanned channels do not cause an error condition and are simply ignored in the evaluation.

When linked as a limit event, a DIO channel will be cleared at the beginning of a new acquisition. Its state will then be updated with each scan according to the programmed limit evaluations. By default, the cleared state is low, but can be set on a per channel basis to be high through the **vtex10xxA_set_dio_limit_event_invert** function. Similarly, the default operation for each channel is non-latch mode, but can be set on a per channel basis to be latch mode through the **vtex10xxA_set_dio_limit_event_latch** function. In latch mode, a transition out of the cleared state would remain, regardless of future limit evaluations, until it is cleared at the beginning of a new acquisition.

The current state of the DIO limit event mechanism is queried with the **vtex10xxA_get_dio_limit_event**, **vtex10xxA_get_dio_limit_event_invert**, and **vtex10xxA_get_dio_limit_event_latch** queries.

CONFIGURE THE TRIGGER MODEL

The EX10xxA supports a full function trigger model with a separate arm source and trigger source event structure. In summary, an acquisition sequence is enabled with a trigger initialize function. Scanning is then initiated upon the receipt of the programmed arm source event followed by the receipt of the programmed trigger source event. Trigger and arm source events can be independently programmed from a variety of sources including Immediate, Timer, Digital I/O, and the Trigger Bus.

Configure the ARM event system

The ARM source is configured with the **vtex10xxA_set_arm_source** function. The ARM event can be specified to be any combination of LXI Trigger Bus channel transitions or levels, Digital I/O channel transitions or levels, LXI LAN event transitions or levels and Timer ticks, or simply be set to Immediate. If multiple sources for an ARM event are specified, they are logically combined as follows:

ARM event = [(Timer tick event) AND (LXI alarm event) AND (LAN event) AND (Digital I/O event) AND (Trigger Bus event)]

Within each digital hardware port, it is also possible to select multiple channels and multiple conditions for a specific channel. In that case, they are logically combined as follows:

Digital I/O event = (Ch 7 events) AND (Ch 6 events) … AND (Ch 0 events)

Finally, within each channel, the four event conditions of Pos. Edge, Neg. Edge, Pos. Level, and Neg. Level are OR'ed together.

As an example, if a Digital I/O event is specified to be a combination of a Pos. Level on channel 3, a Pos. Edge on channel 6, and a Neg. Edge on channel 6, the event will be satisfied when a Pos. Level on channel 3 occurs simultaneously with a Pos. Edge or a Neg. Edge transition on channel 6. While the Digital I/O event structure was used as an example, the Trigger Bus event structure operates in the same manner.

This function accepts the following parameters:

- an array of four 8-bit values representing the enabling of events from any of the 8 channels of the trigger bus. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0.

- an array of four 8-bit values representing the enabling of events from any of the 8 channels of the digital I/O port. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.
- an array of four 8-bit values representing the enabling of events from any of the eight LAN events. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0.
- a Boolean value indicating the enable status of the Timer event
- a Boolean value indicating the enable status of the Immediate event.
- a Boolean value indicating the enable status of the LXI alarm event

Regardless of this setting, software arms are always enabled.

Example #1: This code block enables arm on the timer only.

```
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16 lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 1, 0, 0);
```

Example #2: This code block enables arm on a positive level on DIO channels 0 through 3 and a negative edge on VTB channel 6.

```
ViUInt16 vtb_masks[4] = {0,64,0,0};
ViUInt16 dio_masks[4] = {0,0,0x0F,0};
ViUInt16 lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 0, 0, 0);
```

Example #3: This code block enables software arm only.

```
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 0, 0, 0);
```

The current ARM source is queried with the **vtex10xxA_get_arm_sourceEx** query.

The ARM count is configured with the **vtex10xxA_set_arm_count** function, specified with a value from 1 to (2^{31}-1). This value is reset with each trigger initialize or automatically upon reaching zero when init continuous is enabled.

Example: This code block sets an ARM count of 10.

```
vtex10xxA_set_arm_count(vi, 10);
```

The current ARM count is queried with the **vtex10xxA_get_arm_count** query.

Alternatively, the ARM count can be set to infinity, overriding any manual setting of ARM count, with the **vtex10xxA_set_arm_infinite** function. This function simply accepts a Boolean value indicating the enable status of an infinite ARM count.

The current setting of an infinite ARM count is queried with the **vtex10xxA_get_arm_infinite** query.

The ARM delay is configured with the **vtex10xxA_set_arm_delay** function, specified with a value in seconds from 0 to 4294 with a resolution of 0.000001 (1 µs). The ARM delay is the time between the recognition of the ARM event and the transition into the TRIG layer of the trigger model.

Example: This code block sets an ARM delay of 5 ms.

```
vtex10xxA_set_arm_delay(vi, 0.005);
```

The current ARM delay is queried with the **vtex10xxA_get_arm_delay** query.

Configure the TRIG event system

The TRIG source is configured with the **vtex10xxA_set_trigger_sourceEx** function. The TRIG event can be specified to be any combination of LXI Trigger Bus channel transitions or levels, Digital I/O channel transitions or levels, LXI LAN event transitions or levels and Timer ticks, or simply be set to Immediate. If multiple sources for a TRIG event are specified, they are logically combined as follows:

TRIG event = [(Timer tick event) AND (LXI alarm event) AND (LAN event) AND (Digital I/O event) AND (Trigger Bus event)]

Within each digital hardware port, it is also possible to select multiple channels and multiple conditions for a specific channel. In that case, they are logically combined as follows:

Digital I/O event = (Ch 7 events) AND (Ch 6 events) … AND (Ch 0 events)

Finally, within each channel, the four event conditions of Positive Edge, Negative Edge, Positive Level, and Negative Level are OR'ed together.

As an example, if a Digital I/O event is specified to be a combination of a Pos. Level on channel 3, a Pos. Edge on channel 6, and a Neg. Edge on channel 6, the event will be satisfied when a Pos. Level on channel 3 occurs simultaneously with a Positive Edge or a Negative Edge transition on channel 6. While the Digital I/O event structure was used as an example, the Trigger Bus event structure operates in the same manner.

This function accepts the following parameters:

- an array of four 8-bit values representing the enabling of events from any of the 8 channels of the trigger bus. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0.
- an array of four 8-bit values representing the enabling of events from any of the 8 channels of the digital I/O port. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0.
- an array of four 8-bit values representing the enabling of events from any of the eight LAN events. The order of the values is: positive edge, negative edge, positive level, negative level. Each value is specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0.
- a Boolean value indicating the enable status of the Timer event
- a Boolean value indicating the enable status of the Immediate event.
- a Boolean value indicating the enable status of the LXI alarm event

Regardless of this setting, software triggers are always enabled.

Example #1: This code block enables trigger on the timer only.

```
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16lan_masks[4] = {0,0,0,0};
vtex10xxA_set_trigger_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 1, 0, 0);
```

Example #2: This code block enables trigger on a positive level on DIO channels 0 through 3 and a negative edge on VTB channel 6.

```
ViUInt16 vtb_masks[4] = {0,64,0,0};
ViUInt16 dio_masks[4] = {0,0,0x0F,0};
ViUInt16lan_masks[4] = {0,0,0,0};
vtex10xxA_set_trigger_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 0, 0, 0);
```

Example #3: This code block enables software trigger only.

```
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16lan_masks[4] = {0,0,0,0};
vtex10xxA_set_trigger_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 0, 0, 0);
```

The current TRIG source is queried with the **vtex10xxA_get_trigger_source** query.

The TRIG count is configured with the **vtex10xxA_set_trigger_count** function, specified with a value from 1 to (2^{31}-1). This value is reset with each ARM event.

Example: This code block sets a TRIG count of 10.

```
vtex10xxA_set_trigger_count(vi, 10);
```

The current TRIG count is queried with the **vtex10xxA_get_trigger_count** query.

Alternatively, the TRIG count can be set to infinity, overriding any manual setting of TRIG count, with the **vtex10xxA_set_trigger_infinite** function. This function simply accepts a Boolean value indicating the enable status of an infinite TRIG count.

The current setting of an infinite TRIG count is queried with the **vtex10xxA_get_trigger_infinite** query.

The TRIG delay is configured with the **vtex10xxA_set_trigger_delay** function, specified with a value in seconds from 0 to 4294 with a resolution of 0.000001 (1 µs). The TRIG delay is the time between the recognition of the TRIG event and the execution of the scan list.

Example: This code block sets a TRIG delay of 5 ms.

```
vtex10xxA_set_trigger_delay(vi, 0.005);
```

The current TRIG delay is queried with the **vtex10xxA_get_trigger_delay** query.

Configure general trigger model parameters

The timer interval for Timer source events is configured with the **vtex10xxA_set_trigger_timer** function, specified with a value in seconds from 0.001 (1 ms) to 4294 with a resolution of 0.000001 (1 µs). The timer interval controls the frequency of the Timer event source. The same value is used for both arm and trigger events.

Example: This code block sets a timer interval of 50 ms.

```
vtex10xxA_set_trigger_timer(vi, 0.05);
```

The current timer interval is queried with the **vtex10xxA_get_trigger_timer** query.

The LXI alarm is configured using the **vtex10xxA_self_test**_init function

vtex10xxA_set_alarm_enable call can be performed, **vtex10xxA_set_alarm_enable** must be set to false.

Normally, each acquisition sequence is initiated by a trigger initialize function. However, this requirement can be circumvented through the use of init continuous mode. Specifically, init continuous returns the trigger model to the entrance of the ARM layer without the requirement of a new trigger initialize function. This allows for a specific acquisition sequence to be restarted automatically. The enabling of init continuous mode is done with the **vtex10xxA_set_init_cont** function. This function simply accepts a Boolean value indicating the enable status of init continuous mode.

The current setting of init continuous mode is queried with the **vtex10xxA_get_init_cont** query.

Resetting the trigger model configuration

The **vtex10xxA_reset_trigger_arm** function is used to return all of the EX10xxA's trigger configuration parameters to their default values. It is similar to the **vtex10xxA_reset** function, except that only trigger configuration parameters are affected. These parameters and their reset values are documented in Table 6-5.

NOTE	The extensive flexibility of the trigger model system permits the creation of very specialized trigger conditions, which is a powerful application tool. However, it also permits the creation of trigger conditions that would be very difficult to satisfy in practice. For example, if edge conditions are specified on multiple digital hardware channels, the edges must occur within 25 ns of each other to be recognized as having occurred simultaneously. Similarly, the timer source should not be combined with any digital hardware edge conditions.

INITIATE THE TRIGGER MODEL

A trigger initialize is performed with the **vtex10xxA_init_imm** function. Upon the receipt of this function, the trigger model transitions out of the IDLE layer and begins waiting for the ARM event. This function clears the FIFO reading memory and resets all limit indications.

Once the trigger model has transitioned out of the IDLE layer, most instrument configuration controls are disabled. An acquisition can, however, be aborted at any time with the **vtex10xxA_abort** function. An abort does not affect any stored acquisition data.

CONFIGURE THE LXI TRIGGER BUS

The primary use of the trigger bus is the transmission of high-speed signals for multiple-unit triggering and synchronization. Accordingly, it can be used for input and output operations. Each LXI Trigger Bus (VTB) channel can be independently programmed with regards to its output functionality, its static level to assume when enabled as an output, and pulse operation.

The **vtex10xxA_get_vtb_input** query is used to obtain the measured digital level (0 or 1) of the VTB channels. The input monitoring functionality operates regardless of the simultaneous use of the channels as outputs. The response to this query is a decimal value from 0 to 255 that represents the 8-bit value of the bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0.

Example: This code block queries the state of the trigger bus and reports the level of bits 7, 4, and 0.

```
ViInt32 vtb_in;
vtex10xxA_get_vtb_input(vi, &vtb_in);
if (vtb_in & 0x80)
    printf("Bit 7 is high");
else printf("Bit 7 is low");
if (vtb_in & 0x10)
    printf("Bit 4 is high");
else printf("Bit 4 is low");
if (vtb_in & 0x01)
    printf("Bit 0 is high");
else printf("Bit 0 is low");
```

The **vtex10xxA_set_vtb_output** function is used to set the static level that each channel of the trigger bus will assume if/when enabled as an output. Enabling, however, is done with the **vtex10xxA_set_vtb_output_enable** function (see below). This function accepts a value that represents the desired state of the 8-bit bus, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0.

The **vtex10xxA_set_vtb_output_enable** function enables or disables the output functionality of each channel of the trigger bus. Once enabled, a channel will assume the level specified with the **vtex10xxA_set_vtb_output** function (see above). When not enabled as an output, a channel becomes tri-stated. Input functionality on each channel is constant regardless of its output functionality. This function accepts a value that represents the desired output enable state of the 8-bit bus, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0.

Example: This code block sets bit 7 (high) and bit 6 (low) of the trigger bus and then enables them as outputs.

```
vtex10xxA_set_vtb_output(vi, 0x80);
vtex10xxA_set_vtb_output_enable(vi, 0xC0);
```

The **vtex10xxA_get_vtb_output** query is used to obtain the programmed output state of the trigger bus. The response to this query is a decimal value from 0 to 255 that represents the programmed output state of the 8-bit bus. However, since the outputs must be enabled, it does not necessarily represent the true output state. The **vtex10xxA_get_vtb_output_enable** query is used to obtain the output enable state of each channel. The response to this query is a decimal value from 0 to 255 that represents the output enable state of the 8-bit bus. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0.

Example: This code block queries the output state and output enable state of the trigger bus and reports the states of bit 4.

```
ViInt32 vtb_out;
ViInt32 vtb_outen;
vtex10xxA_get_vtb_output(vi, &vtb_out);
vtex10xxA_get_vtb_output_enable(vi, &vtb_outen);
if (vtb_out & 0x10)
    printf("Bit 4 is set high");
else printf("Bit 4 is set low");
if (vtb_outen & 0x10)
    printf("Bit 4 is enabled");
else printf("Bit 4 is not enabled");
```

The **vtex10xxA_set_vtb_pulse** function is used to generate a 1 μs pulse on selected channels of the trigger bus. The pulse will occur only if the selected channels are enabled as outputs. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is

programmed with a static level of low, the pulse will be high-going. Each pulse generation requires a separate function. This function accepts a value that represents the channels to be pulsed within the 8-bit bus, specified in either decimal (0 through 255) or hex (0x00 through 0xFF). Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0.

Example: This code block sets bit 7 to a static level of low and then generates a high-going pulse.

```
vtex10xxA_set_vtb_output(vi, 0x00);
vtex10xxA_set_vtb_output_enable(vi, 0x80);
vtex10xxA_set_vtb_pulse(vi, 0x80);
```

TRIGGER EVENT

An acquisition sequence will be conducted once the programmed arm and trigger events are recognized. In addition to the programmed event sources, software arms and software triggers are always enabled. These are used to put the trigger model under direct software program control.

A software arm is performed by the **vtex10xxA_soft_arm** function. The software arm applies at the time it is issued and will be ignored if the system is not waiting for an ARM event. The issuance of a software arm has no effect on the ARM source configuration.

A software trigger is performed by the **vtex10xxA_soft_trigger** function. The software trigger applies at the time it is issued and will be ignored if the system is not waiting for a TRIG event. The issuance of a software trigger has no effect on the TRIG source configuration.

RETRIEVING DATA (READ FIFO AND STREAMING DATA)

The EX10xxA stores acquisition data in a large, on-board FIFO in the instruments RAM memory. 14 MB of RAM are reserved for the on-board FIFO. There are two primary mechanisms for retrieving acquisition data from the EX10xxA FIFO:

- Read FIFO
- Asynchronous Streaming Data

The Read FIFO mechanism is similar to the way data is returned from traditional data acquisition instruments, with the user application making periodic (polling) queries of the instrument to retrieve data from the instrument's on-board FIFO, while asynchronous data streaming is a more modern, efficient technique, made possible by the instrument's LXI interface, in which the instrument automatically transmits acquisition data to the user application as data becomes available.

The streaming data interface is slightly more complicated to use than the Read FIFO interface, but makes very efficient use of the host PC's processor and the test system's network. As such, the streaming data interface scales well for high channel count and/or high sample rate systems. The two data retrieval mechanisms are mutually exclusive – if the streaming data interface is enabled, Read FIFO requests will return an error.

Read FIFO

The EX10xxA utilizes a 14 MB FIFO memory storage to buffer acquisition data prior to retrieval. The amount of scans that can be buffered within the memory is dependent on the number of channels in the scan list and the requested data format. See *Acquiring Data* in Section 3 for specific formulas and examples. The number of data pages (scans) that have yet to be retrieved from FIFO memory is obtained using the **vtex10xxA_get_fifo_count** function.

Figure 6-11 shows the general sequence of events when using the **vtex10xxA_read_fifo** or **vtex10xxA_read_fifoEx** functions. In this diagram, time flows from top to bottom. The two vertical lines represent the two network nodes: the host PC, running the user application, and the EX10xxA instrument. The diagonal arrows connecting them represent network messages sent between them

(the diagonal arrow, instead of a horizontal arrow, indicates that the message is not received instantaneously). As illustrated in this diagram, each instrument driver function call results in two network messages: one to the EX10xxA (a request) and one from the EX10xxA (a response). The instrument driver function does not return control to the user application until the response message is received.

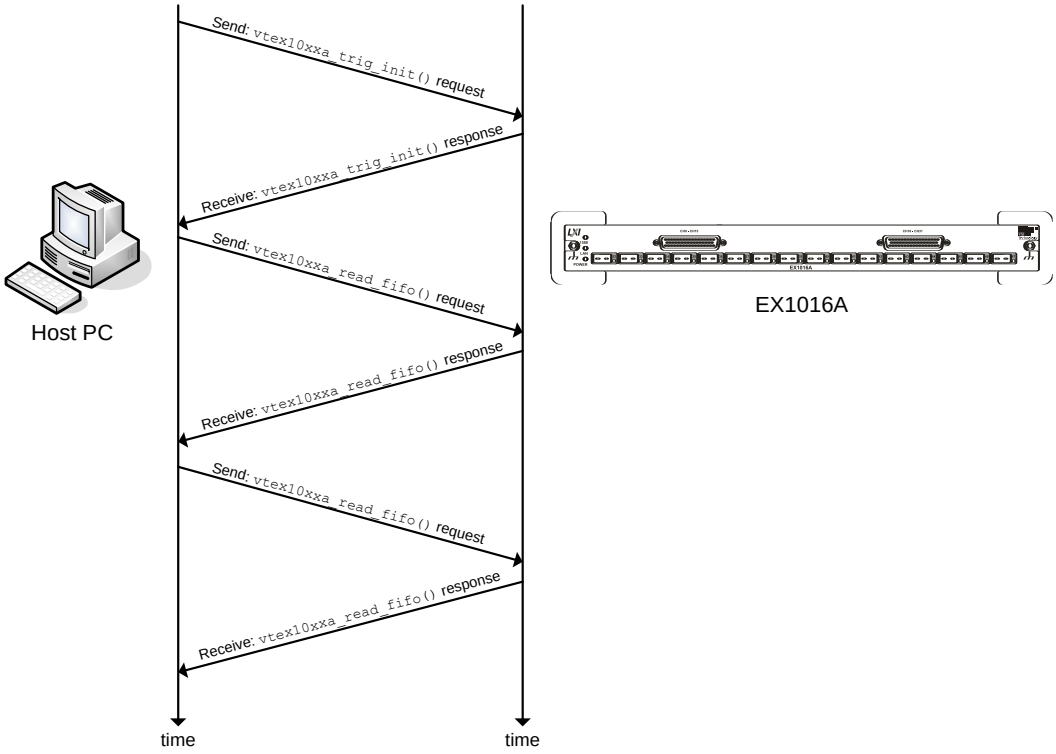


FIGURE 6-10: READ FIFO NETWORK EXAMPLE

As can be seen by the vertical distances, each of the instrument driver function calls takes some finite amount of time, allowing for host PC processing, network transmission and reception, and instrument processing. For instrument setup (e.g., configuring acquisition channels or trigger parameters), this time is typically negligible. For retrieving acquisition data, however, these delays can become significant. This is especially true in high sample count and/or high channel count systems. In such systems, the host PC can waste a significant amount of CPU time in these polling loops, also consuming network bandwidth. The streaming data mechanism offers a more efficient alternative.

Acquisition data is retrieved from the EX10xxA using the **vtex10xxA_read_fifo** or **vtex10xxA_read_fifoEx** functions. The parameters of these functions control the number of data scans performed and the allowed time during which data retrieval is attempted. Data returned by both functions include input channel measurement data and the start time of each scan. The **vtex10xxA_read_fifoEx** function also returns CJC channel measurement data and individual channel timestamp data.

The syntax of this function is complex, and it is important for the user to understand the application of its various input parameters and return values. Another important concept is that this function is a blocking function. That is, once it is sent by an application, no further functions can be executed within that application until the **vtex10xxA_read_fifo** or **vtex10xxA_read_fifoEx** function has completed its execution.

There are two ways for the execution to complete. The most obvious way is for the requested number of scans to actually be delivered by the EX10xxA. This is considered to be successful execution.

The other way is for a specified timeout period to elapse. In this case, the full number of requested scans was not delivered; this is considered an execution error. However, there are applications where this is the expected and desired behavior, and all data returned is completely valid, despite this being technically an error condition. These concepts are clarified below.

This function accepts the following parameters:

- the maximum number of scans to return (*maxscans*). This represents the desired number of scans to retrieve. If FIFO count ≥ *maxscans* at the time of function issuance, then *maxscans* are returned, and the function completes. If, however, FIFO count < *maxscans* at the time of function issuance, the function will continue to poll the EX10xxA for data until *maxscans* are returned or the timeout period (*to_secs*) is reached (see below).
- a return array of scan start times (*ts_secs[]*), specified in seconds since the epoch (Jan. 1, 1970). To ensure data integrity, the declared dimension of *ts_secs[]* must be at least as large as *maxscans*.
- a return array of scan start times (*ts_fsecs[]*), specified in seconds since the last full second represented in *ts_secs[]*. To ensure data integrity, the declared dimension of *ts_fsecs[]* must be at least as large as *maxscans*.
- a return value indicating the actual number of scans retrieved (*numscans*). If the function completed by reaching *maxscans*, then *numscans* will be equal to *maxscans*. If, however, the function completed by the timeout period, *numscans* will be less than *maxscans*.
- the maximum length of the return data array (*maxdata*). This should specifically be the declared dimension of the *data[]* array (see below). To ensure data integrity, *maxdata* must be no less than *maxscans* multiplied by the number of channels in the scan list. Since data is retrieved from the FIFO according to the *maxscans* parameter and not the *maxdata* parameter, if *maxdata* is set too low, there is the possibility of data being permanently lost.
- a return array of sample data (*data[]*).
- a return value indicating the actual number of samples retrieved (*numdata)*. This will be equal to the lesser of *maxdata* or *numscans* multiplied by the number of channels in the scan list.
- a timeout period in seconds (*to_secs*) that represents how long to poll the EX10xxA for data (in an attempt to retrieve *maxscans*). A value of 0 is used to indicate an infinite timeout period.

The importance of the *to_secs* parameter depends on the trigger model configuration employed. In a typical application where the ARM source is set to Immediate and the TRIG source is set to Timer, the amount of data to be acquired and the time required to acquire that data are deterministic. Specifically, upon the trigger initialize function, there will be a quantity of scans acquired that equals TRIG count, acquired over a total time that is TRIG count multiplied by the timer interval. For example, 500 scans taken with a timer interval of 5 ms would require 2.5 s. In this case, *to_secs* would normally be set to an arbitrary value greater than 2.5. In this case, function completion via the timeout period does indeed represent an error condition.

By contrast, consider an application where data is desired from the EX10xxA only once a certain test condition has occurred. Accordingly, the ARM source is set to a specific condition of the digital I/O port. Moreover, this digital event is created asynchronously by a stimulus system being controlled by the same application that is taking EX10xxA data. The logic of the control application is such that the stimulus system is increased to a new value and then the EX10xxA is polled for data. If data is returned, then the EX10xxA was triggered, and no further increase in the stimulus system is required. If no data is returned after a specified period, then the stimulus system is increased further. In this case, the timeout period value is important, as it determines the frequency at which the stimulus system is updated. Moreover, it is completely expected that the **vtex10xxA_read_fifo** function initially completes due to the timeout period, and so this does not represent an error condition that requires intervention.

Example: This code block reads the FIFO after a timer-based scan.

```
#define NUM_CHANNELS        5
#define MAX_SCANS           20
#define MAX_DATA            (NUM_CHANNELS * MAX_SCANS)
#define TIMER_INTERVAL      0.01

ViInt32    channels[NUM_CHANNELS] = {0, 1, 2, 3, 4};
ViReal64 ts_secs[MAX_SCANS], ts_fsecs[MAX_SCANS];
ViInt32    num_scans;
ViReal64 data[MAX_DATA];
ViInt32    num_data;

// set the scanlist
vtex10xxA_set_scanlist(vi, channels, NUM_CHANNELS);

// set the trigger source to be timer
vtex10xxA_set_trig_source_timer(vi, TIMER_INTERVAL);

// set trigger count
vtex10xxA_set_trigger_count(vi, MAX_SCANS);

// trigger initialize
vtex10xxA_init_imm(vi);

// retrieve the data, nominally ready in 2 seconds
vtex10xxA_read_fifo(vi, MAX_SCANS, ts_secs, ts_fsecs, &num_scans, MAX_DATA, data,
&num_data, 3);
```

Data correlation

The data returned by the **vtex10xxA_read_fifo** function is written into three one-dimensional arrays, termed *data[]*, *ts_secs[]*, and *ts_fsecs[]* in this example. For a five-channel scan, the data from the first scan will be in elements data[0] to data[4] with a start of scan time indicated by ts_secs[0] + ts_fsecs[0]. Data from the second scan will be in elements data[5] to data[9] with a start of scan time indicated by ts_secs[1] + ts_fsecs[1], and so on. Within each scan, the first data element represents the first declared entry in the scan list. The second data element represents the second declared entry in the scan list, and so on.

NOTE	In order to provide the maximum reading buffer capacity for future acquisitions, data is deleted from the FIFO memory upon retrieval.

Asynchronous Streaming Data

The asynchronous streaming data interface optimizes communication between the host PC and the EX10xxA. The asynchronous streaming data interface allows the EX10xxA to transmit acquisition data to the host PC whenever data is available. It "streams" data to the host PC – that is the EX10xxA transmits data when available – and is "asynchronous" in that data arrives outside the normal control flow of the user-application. This is in contrast to the Read FIFO mechanisms, where the client polls or queries the instrument for data, and the data is returned to the user application when the **vtex10xxA_read_fifo**/**vtex10xxA_read_fifoEx** function returns (synchronous with the normal program control flow).

Figure 6-11 illustrates the general sequence of events for the streaming data mechanism. This can be compared to Figure 6-10 for using the Read FIFO mechanism. Prior to initiating the acquisition (**vtex10xxA_init**), the streaming data interface must be enabled via the **vtex10xxA_enable_streaming_data** or **vtex10xxA_enable_streaming_dataEx** functions. This configures the streaming data communication on both the instrument and the host PC. It is important that the streaming data interface be enabled prior to initiating acquisition, as the EX10xxA prevents streaming data from being enabled after initiating acquisition. As Figure 6-11 shows, the EX10xxA

transmits acquisition data to the host PC periodically, whenever data is available, without the host having to request it.

The streaming data interface uses a separate “socket”, or communications link, than the one used for other instrument driver functions. Since TCP/IP can support thousands of concurrent sockets, all multiplexed on the same network interface, this does not present a problem for the network.

NOTE	The network communication diagrams provided are oversimplifications. Since TCP/IP is used as the transport layer, there are potentially several Ethernet packets involved (send and receive) in each high-level message. These packets provide, among other things, the reliable data transport feature of TCP.

The asynchronous nature of the streaming data arrival at the host PC presents the issue of how to deliver the data to the user application. For efficiency, particularly when the acquisition system consists of many instruments, a multi-threaded model was chosen.

Multi-threaded programming is beyond the scope of this manual, but the general idea is that an application can have multiple, concurrent “threads” of control. By default, all applications have one thread, the one that begins executing at the main() function (or similar entry point, depending on the programming language). Optionally, applications may have additional, programmer created threads. These threads all execute in the same memory space, making it very efficient for them to share data. This is different from multi-process programming, wherein each process – basically a memory space with a single, default thread – executes independently.

Threads execute asynchronously to each other by default – that is, their execution relative to other threads within the same application (process) is non-deterministic, and shared data must be protected by design or through suitable inter-thread communication mechanisms (e.g., mutexes) to guarantee consistency. Again, multi-threaded programming is beyond the scope of this manual, but it is important to understand the fundamentals before the streaming data mechanism can be used properly. For more information on this topic, we recommend reviewing a textbook on operating systems (e.g., *Operating System Concepts*, by Silberschatz, Galvin, and Gagne or *Modern Operating Systems*, by Tanenbaum) as well as the Windows SDK information available online.

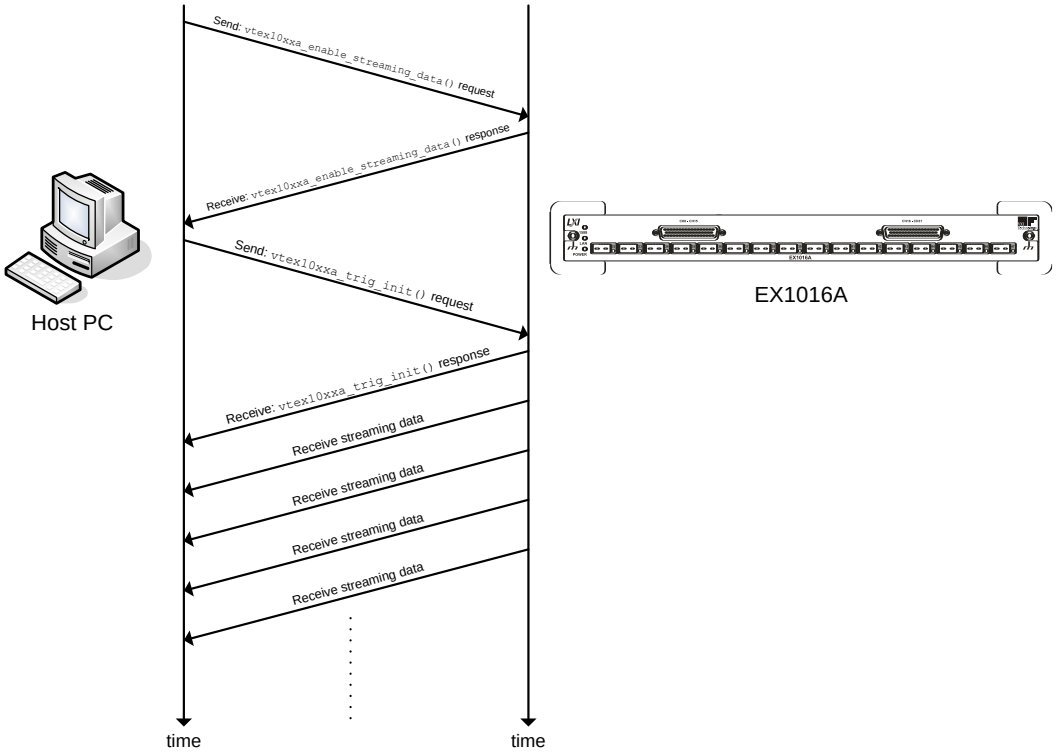


FIGURE 6-11: STREAMING DATA NETWORK EXAMPLE

Basic Streaming Data Usage

When using the streaming data interface, via the **vtex10xxA_enable_streaming_data** function, the user application provides a callback function. Internally, the instrument driver creates a thread and then opens a socket for streaming data between the host PC and the instrument. The newly constructed thread does a "blocking" read on the socket, which causes it to "sleep" (become idle) until data arrives. When acquisition data arrives, the thread begins executing, receives the acquisition data from the instrument, executes the user-provided callback function, passing in the newly arrived data, and then returns to the "sleep" state. The callback function can do whatever is necessary for the application: write the acquisition values to a file on disk, perform limit checking on the acquisition values, update an application-specific data structure (e.g., FIFO) post the acquisition data to a database or spreadsheet, etc. This behavior is illustrated in Figure 6-12, with the reception of streaming data causing the user-provided callback function to be executed.

NOTE	Since the callback function executes asynchronously in the same process as the main application thread, it is important that any data or data structures used by both threads are suitably protected to guarantee consistency. As with any multi-thread application, care must be taken when using inter-thread communication primitives (e.g., mutexes) to prevent deadlocks and livelocks. Similarly, performing GUI operations (e.g., updating an on-screen graph) within the callback function needs to be implemented carefully.

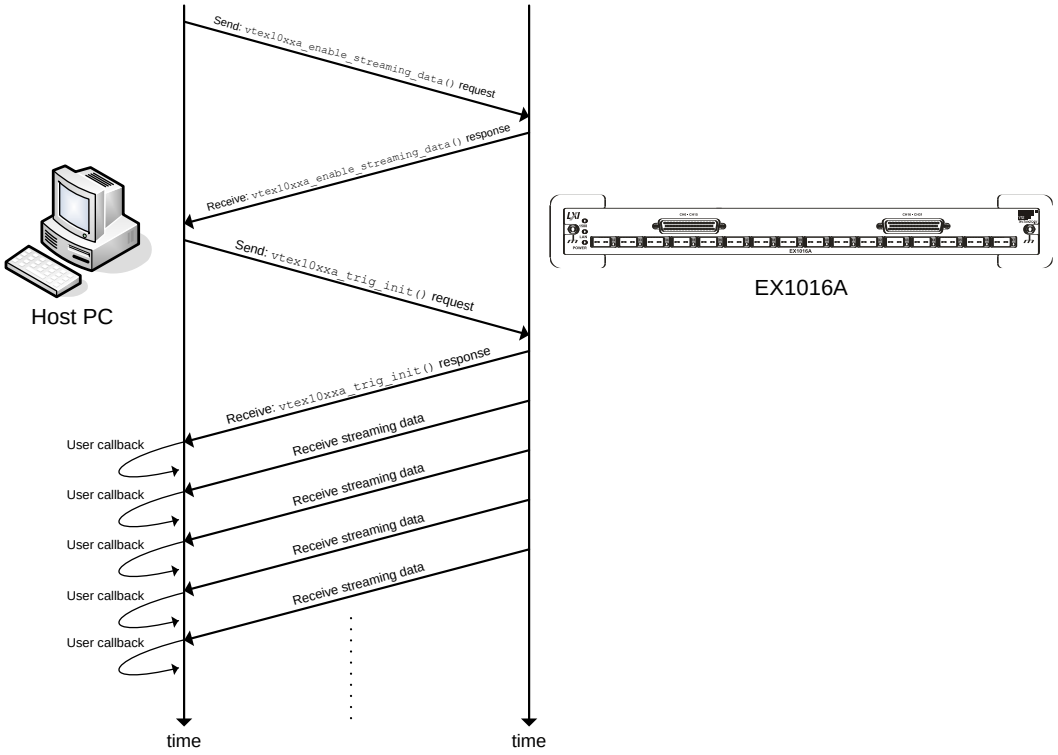


FIGURE 6-12: STREAMING DATA WITH USER CALLBACK

The following sample code segment illustrates a very basic use of the streaming data interface. The callback function, stream_callback, just prints the timestamps and data values to a FILE handle. The FILE handle, as well as a sample count total variable, are stored in a user-defined data structure. A pointer to this structure is passed to the **vtex10xxA_enable_streaming_data** function, along with a function pointer to the streaming callback function. Later, when streaming data pages (scans) are received, a pointer to the acquisition data, along with the pointer to the user-defined data structure, are passed to the callback function. The user-defined data structure pointer is passed as a void* and should be cast to the appropriate type within the callback function.

Streaming Data Example Program

```
// INCLUDE FILES & DEFINES

#define INSTR_LANGUAGE_SPECIFIC
#include <vtex10xxA.h>
#include <visa.h>
#include <vpptype.h>
#include <visatype.h>


#define RESOURCE_10XX          "TCPIP0::192.168.1.103::INSTR"


ViSession ex10XXHandle = 0;
ViStatus error = VI_SUCCESS


ViInt32 streaming_function( void *priv, EX10xxA_SampleData *data );
```

```
// MAIN EXECUTION START

int main (int argc, char *argv[])
{

	ViInt32 channels[48], numChannels = 48, channelCounter = 0;
	ViInt32 user_data = 0;

	error = vtex10xxA_init (RESOURCE_10XX, VI_TRUE, VI_TRUE, &ex10XXHandle);

	for(channelCounter = 0; channelCounter < numChannels; channelCounter++)
	{
		channels[channelCounter] = channelCounter;
	}

	//Set the scanlist
	error = vtex10xxA_set_scanlist(ex10XXHandle, channels, numChannels);

	//Set the channel conversion to volts
	error = vtex10xxA_set_channel_conversion(ex10XXHandle, channels, numChannels, VTEX10XXA_CONV_MV);

	//Enable the streaming interface
	error = vtex10xxA_enable_streaming_data(ex10XXHandle, &user_data, streaming_function);

	//Set the trigger source to timer
	error = vtex10xxA_set_trig_source_timer(ex10XXHandle, 0.01);

	//Set the trigger count to infinite for continuous acquisition
	error = vtex10xxA_set_trigger_infinite(ex10XXHandle, VI_TRUE);

	//Enable usage of custom limit values
	error = vtex10xxA_set_limit_set0_manual(ex10XXHandle, channels, numChannels, VI_TRUE);

	//Set up custom limit values
	error = vtex10xxA_set_limit_set0(ex10XXHandle, channels, numChannels, -2.0, 2.0);

	//Initialize voltage acquisition
	error = vtex10xxA_init_imm(ex10XXHandle);


	//At this point the program will loop through the streaming callback forever since the trigger is set to infinite

	//Wait for some stop condition to occur


	//Abort acquisition
	error = vtex10xxA_abort(ex10XXHandle);

	//disable the streaming interface
	error = vtex10xxA_disable_streaming_data(ex10XXHandle);

	return 0;
}



// STREAMING DATA CALLBACK
```

```
ViInt32  streaming_function( void *priv, EX10xxA_SampleData *data )
{
	int channel;
	int limits_broken = 0;

	//Check limit status

	for (channel = 0; channel < data->x.x_len; channel++)
	{
		int offset = channel / 32;
		int mask = 1 << (channel % 32);

		if ( (data->limits.limits_val[0+offset] & mask) != 0 ||
		    (data->limits.limits_val[2+offset] & mask) != 0 )
		{
			//Limits have been breached for this channel.
		}
	}

	//Print the data

	printf("%u.%09u", data->ss_secs, data->ss_nsecs);
	for (channel = 0; channel < data->x.x_len; channel++)
	{
		printf(", %f", data->x.x_val[channel]);
	}
	printf("\n");

	return 0;
}
```

It is imperative that the streaming data interface be enabled prior to initializing acquisition (the **vtex10xxA_init** function) and disabled after acquisition completes, or is aborted explicitly (**vtex10xxA_abort**).

The streaming callback function extracts the acquisition and timestamp data from the EX10xxA_SampleData structure. The EX10xxA_SampleData structure is documented in the instrument driver header file and can also be found in the **vtex10xxA_enable_streaming_data**.

EXAMPLE PROGRAM

```
#include <stdio.h>
#include <windows.h>
#include <vtex10xxA.h>

#define INSTR_RESRC_STR"TCPIP::192.168.0.127::INSTR"
#define TYPE_V                  VTEX10XXA_CONV_MV
#define TYPE_T                  VTEX10XXA_CONV_THERMO_TYPE_T
#define TYPE_E                  VTEX10XXA_CONV_THERMO_TYPE_E
#define NUM_CHANNELS            11
#define NUM_V_CHANNELS          2
#define NUM_E_CHANNELS          4
#define NUM_T_CHANNELS          5
#define TRIG_TIMER              0.2         /* (5 readings per sec) */
#define MAX_SCANS               1000
#define MAX_DATA                (NUM_CHANNELS * MAX_SCANS)

int main( int argc, char **argv )
{
   ViSession        vi;
   ViStatus         status;
   /* scanlist entries for EX1032A */
   /* channel 0 - 15: voltage channels, 16 - 47: thermocouple channels */
   ViInt32          channels[NUM_CHANNELS] = {0,1,3,6,10,15,24,30,31,32,35};
   ViInt32          v_channels[NUM_V_CHANNELS] = {0,1};
   ViInt32          e_channels[NUM_E_CHANNELS] = {3,6,10,15};
   ViInt32          t_channels[NUM_T_CHANNELS] = {24,30,31,32,35};
   ViInt32          i, j;
   ViReal64         ts_secs[MAX_SCANS], ts_fsecs[MAX_SCANS];
   ViInt32          numscans;
   ViReal64         data[MAX_DATA];
   ViInt32          numdata;

   /* open a session */
   status = vtex10xxA_init(INSTR_RESRC_STR, VI_ON, VI_ON, &vi);
   if(status != VI_SUCCESS)
   {
      printf("ERROR OPENING CONNECTION\n");
      return -1;
   }
   printf("Connection opened to %s\n", INSTR_RESRC_STR);

   /* configure the scan list */
   status = vtex10xxA_set_scanlist(vi, channels, NUM_CHANNELS);
   if(status != VI_SUCCESS)
   {
      printf("ERROR CONFIGURING SCAN LIST\n");
      return -1;
   }

   /* configure the EU conversions */
   status = vtex10xxA_set_channel_conversion(vi, v_channels, NUM_V_CHANNELS, TYPE_V);
   if(status != VI_SUCCESS)
   {
      printf("ERROR CONFIGURING EU CONVERSIONS\n");
      return -1;
```

```
}
status = vtex10xxA_set_channel_conversion(vi, e_channels, NUM_E_CHANNELS, TYPE_E);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING EU CONVERSIONS\n");
   return -1;
}
status = vtex10xxA_set_channel_conversion(vi, t_channels, NUM_T_CHANNELS, TYPE_T);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING EU CONVERSIONS\n");
   return -1;
}

/* configure the filter frequencies */
status = vtex10xxA_set_filt_freq(vi, channels, NUM_CHANNELS, 4.0);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING FILTER FREQUENCIES\n");
   return -1;
}

/* configure the voltage channel range */
status = vtex10xxA_set_channel_range(vi, v_channels, NUM_V_CHANNELS, 10.0);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING CHANNEL RANGE\n");
   return -1;
}

   /* configure the FIFO (deg F, blocking mode) */
status = vtex10xxA_set_fifo_config(vi, 0, 0, 0, 1);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING FIFO\n");
   return -1;
}

/* configure the trigger model */

/* reset the trigger model to default settings */
status = vtex10xxA_reset_trigger_arm(vi);

/* set the trigger timer */
status = vtex10xxA_set_trigger_timer(vi, TRIG_TIMER);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING TIMER\n");
   return -1;
}

/* set the trigger count */
status = vtex10xxA_set_trigger_count(vi, MAX_SCANS);
if(status != VI_SUCCESS)
{
   printf("ERROR CONFIGURING COUNT\n");
   return -1;
```

```
    }

    /* initialize the acquisition */
    status = vtex10xxA_init_imm(vi);
    if(status != VI_SUCCESS)
    {
        printf("ERROR INITIATING TRIGGER\n");
        return -1;
    }

    /* read acquisition data */
    status = vtex10xxA_read_fifo(vi, MAX_SCANS, ts_secs, ts_fsecs, &numscans, MAX_DATA, data, &numdata, (ViInt32)
(MAX_SCANS * TRIG_TIMER + 0.1));
    if(status != VI_SUCCESS)
    {
        printf("ERROR READING DATA\n");
        return -1;
    }

    /* print acquisition data */
    for(i = 0; i < numscans; i++)
    {
        printf("%.0f.%09.0f: ", ts_secs[i], ts_fsecs[i] * 1e9);
        for(j = 0; j < NUM_CHANNELS; j++)
        {
            printf("%6.2f ", data[i * NUM_CHANNELS + j]);
        }
        printf("\n");
    }

    /* close the session */
    status = vtex10xxA_close(vi);

    return 0;
}
```

SECTION 7

FUNCTION CALLS

INTRODUCTION

This section presents the instrument function set. It begins by listing the APIs according to function and is then followed by an alphabetical listing. With each function is a brief description. The remainder of this section is devoted to describing each function in detail. Each function entry provides the function prototype, the use and range of parameters, and a description of the function's purpose.

Function Return Value

Each function will return a status that will contain either VI_SUCCESS or an error status returned by the function call.

Error Messages section found later in the chapter for possible error codes. If the vtex10xxA_error_message function call is used, it will return a description of the error code returned by the last function call made.

ALPHABETICAL FUNCTION SET

The following table provides a summary of the APIs used by the EX10xxA along with an abbreviated description of the function. The pages following this table are definitions that provide in-depth detail for each API. A sample API definition is provided immediately following this table to illustrate what each section of the definition describes.

API	Description
vtex10xxA_abort	Aborts the current acquisition.
vtex10xxA_append_scanlist	Adds channels onto the end of an existing scan list.
vtex10xxA_break_lock	Breaks a lock on the instrument.
vtex10xxA_check_lock	Returns the lock status of the instrument.
vtex10xxA_clear_lan_eventlog	Clears event log entries in the instrument.
vtex10xxA_close	Closes an instrument programming session.
vtex10xxA_disable_streaming_data	Disables data streaming.
vtex10xxA_enable_streaming_data	Enables data streaming.
vtex10xxA_enable_streaming_dataEx	Enables data streaming (compatible with EX1048 function calls).
vtex10xxA_error_message	Outputs the error message associated with the **errCode** parameter.
vtex10xxA_error_query	Outputs the error code and error message obtained from the instrument.
vtex10xxA_get_accum_limit_status	Returns the accumulated limit status of all 48 channels.
vtex10xxA_get_alarm	Returns the LXI alarm, time, interval and count for the alarm source event.
vtex10xxA_get_alarm_enable	Returns the enabled status of the LXI alarm
vtex10xxA_get_arm_count	Returns the arm count value.
vtex10xxA_get_arm_delay	Returns the arm delay.

API	Description
vtex10xxA_get_arm_infinite	Returns the enabled status of an infinite arm count.
vtex10xxA_get_arm_lan_eventID	Returns the event ID of a LAN arm event.
vtex10xxA_get_arm_lan_filter	Returns the filter for the LAN arm event.
vtex10xxA_get_arm_source	Returns the enabled arm source events.
vtex10xxA_get_arm_sourceEx	Returns the arm source events including LAN events.
vtex10xxA_get_calibration_file	Returns the calibration file data for the specified buffer.
vtex10xxA_get_calibration_running	Indicates if calibration is currently in progress.
vtex10xxA_get_channel_conversion	Returns the engineering units (EU) conversion of a specified channel.
vtex10xxA_get_channel_range	Returns the input range of a voltage channel.
vtex10xxA_get_channel_type	Returns whether specified channel is voltage or thermocouple only channel.
vtex10xxA_get_dio_input	Returns the current input state of the digital I/O port.
vtex10xxA_get_dio_limit_event	Returns the enabled DIO limit events.
vtex10xxA_get_dio_limit_event_invert	Returns the inverted operation of a DIO channel linked as a limit event.
vtex10xxA_get_dio_limit_event_latch	Returns the latch operation of a DIO channel linked as a limit event.
vtex10xxA_get_dio_output	Returns the programmed output state of the digital I/O port.
vtex10xxA_get_dio_output_enable	Returns the output enable state of the digital I/O port.
vtex10xxA_get_fifo_config	Returns the data format and overflow behavior of the FIFO memory.
vtex10xxA_get_fifo_count	Returns the number of data pages (scans) in the FIFO memory.
vtex10xxA_get_filt_freq	Returns current low-pass filter cutoff frequency of a specified channel.
vtex10xxA_get_init_cont	Returns the enabled status of init continuous mode.
vtex10xxA_get_lan_event_domain	Returns the LAN domain for the instrument.
vtex10xxA_get_lan_event_source_state	Returns the current arm and trigger state of the LAN events.
vtex10xxA_get_lan_eventlog_count	Returns the number of event log entries logged in the instrument.
vtex10xxA_get_lan_eventlog_enabled	Returns the current status of the event log.
vtex10xxA_get_lan_eventlog_overflowmode	Returns the LAN event log overflow mode.
vtex10xxA_get_limit_set0	Returns the limit set 0 values of a specified channel.
vtex10xxA_get_limit_set0_manual	Returns the manual entry control of limit set 0 values of a specified channel.
vtex10xxA_get_limit_set1	Returns the limit set 1 values of a specified channel.
vtex10xxA_get_linear_correction	Returns linear correction of a voltage channel.
vtex10xxA_get_model	Returns the instrument's model name.
vtex10xxA_get_OTD_enable	Returns a channel's open transducer detection status.
vtex10xxA_get_ptp_info	Returns the instrument's PTP master and synchronization information.
vtex10xxA_get_scanlist	Returns the current scan list.
vtex10xxA_get_self_test_result	Returns the self test result (Currently supported only in RX10xx)
vtex10xxA_get_self_test_running	Returns the running status of self test result. (Currently supported only in RX10xx)
vtex10xxA_get_serialNumber	Returns the serial number of the device.
vtex10xxA_get_system_time	Returns the instrument's current time in seconds.
vtex10xxA_get_trig_lan_eventID	Returns the event ID of a LAN trigger event.
vtex10xxA_get_trig_lan_filter	Returns the filter for a LAN trigger event.
vtex10xxA_get_trigger_count	Returns the trigger count value.
vtex10xxA_get_trigger_delay	Returns the trigger delay.
vtex10xxA_get_trigger_infinite	Returns the enabled status of an infinite trigger count.
vtex10xxA_get_trigger_source	Returns the trigger source excluding LAN events.
vtex10xxA_get_trigger_sourceEx	Returns the trigger source including LAN events.
vtex10xxA_get_trigger_timer	Queries the timer interval for the timer source event.
vtex10xxA_get_user_cjc_enable	Returns the enabled status of a user-defined CJC temperature of a specified channel.
vtex10xxA_get_user_cjc_temp	Returns the user-defined CJC temperature of a specified channel.

API	Description
vtex10xxA_get_user_conversion	Returns the user-defined conversion polynomials.
vtex10xxA_get_vtb_input	Returns the current input state of the trigger bus.
vtex10xxA_get_vtb_output	Returns the programmed output state of the trigger bus.
vtex10xxA_get_vtb_output_enable	Returns the output enable state of the trigger bus.
vtex10xxA_get_vtb_wiredor	Returns the Wired-OR state for each LXI Trigger Bus channel.
vtex10xxA_get_vtb_wiredor_bias	Returns the Wired-OR bias enable state for each LXI trigger bus channel.
vtex10xxA_init	Opens an instrument programming session.
vtex10xxA_init_imm	Performs a trigger initialize.
vtex10xxA_lock	Attempts to acquire a lock on the instrument.
vtex10xxA_pop_logged_LAN_event	Returns an oldest event log entry in the instrument.
vtex10xxA_read_fifo	Retrieves acquisition data.
vtex10xxA_read_fifoEx	Retrieves acquisition data with timestamps and CJC data.
vtex10xxA_reset	Performs an instrument reset.
vtex10xxA_reset_fifo	Clears the FIFO memory.
vtex10xxA_reset_trigger_arm	Performs a reset of the trigger configuration parameters.
vtex10xxA_revision_query	Returns the release revision of the instrument driver and embedded firmware.
vtex10xxA_self_cal_clear	Clears the current self cal data.
vtex10xxA_self_cal_clear_stored	Clears self cal data from nonvolatile memory.
vtex10xxA_self_cal_get_status	Returns the completion status of self-calibration.
vtex10xxA_self_cal_init	Performs an instrument self-calibration.
vtex10xxA_self_cal_is_stored	Returns the presence of self cal data in nonvolatile memory.
vtex10xxA_self_cal_load	Loads nonvolatile self cal data as the current self cal data.
vtex10xxA_self_cal_store	Stores the current self cal data into nonvolatile memory.
vtex10xxA_self_test	Performs a self-test in **synchronous** mode. (Currently supported only in RX10xx.)
vtex10xxA_self_test_init	Performs a self test operation in **asynchronous** mode. (Currently supported only in RX10xx.)
vtex10xxA_set_alarm	Sets the alarm time, period, and count for the LXI alarm source event.
vtex10xxA_set_alarm_enable	Enables or disables the use of the LXI alarm.
vtex10xxA_set_arm_count	Sets the arm count value.
vtex10xxA_set_arm_delay	Sets the arm delay.
vtex10xxA_set_arm_infinite	Enables or disables the use of an infinite arm count.
vtex10xxA_set_arm_lan_eventID	Sets the event ID of LAN arm event.
vtex10xxA_set_arm_lan_filter	Sets the filter for the LAN arm event.
vtex10xxA_set_arm_source	Sets the arm source. Excludes LAN events.
vtex10xxA_set_arm_sourceEx	Sets the arm source. Includes LAN events.
vtex10xxA_set_channel_conversion	Sets the engineering units (EU) conversion for the specified channels.
vtex10xxA_set_channel_range	Sets the input range of a voltage channel.

API	Description

vtex10xxA_set_communication_**timeout**
Function Prototype
ViStatus _VI_FUNC
vtex10xxA_set_communication_timeout
(ViSession vi, ViInt32 timeout);
Function Parameters
`vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.`
SETS THE TIMEOUT VALUE (IN MILLISECONDS). INDICATING HOW LONG TO WAIT FOR INSTRUMENT TO RESPOND. IN HIGHER LATENCY NETWORK, IT IS BETTER NOT TO REDUCE DEFAULT TIMEOUT VALUE. VALID RETURN VALUES: 1000 THROUGH (231-1). DEFAULT VALUE: 25000 (25 SECONDS).
Data Item Reset Value
Reset will not change value of this parameter.
Description
This function sets the timeout value when a LAN communication error occurs. It is recommended for advanced users only.
Example
//sets the LAN communication timeout to 10 seconds, if a LAN communication error occurs
vtex10xxA_set_communication_**timeout(vi, 10000);**
	Links limit evaluations to the operation of the digital I/O port.

vtex10xxA_set_dio_limit_event	
vtex10xxA_set_dio_limit_event_invert	Enables or disables inverted operation of a DIO channel linked as a limit event.
vtex10xxA_set_dio_limit_event_latch	Enables or disables latch operation of a DIO channel linked as a limit event.
vtex10xxA_set_dio_output	Sets the static level that each channel of the digital I/O port will assume if enabled.
vtex10xxA_set_dio_output_enable	Enables or /disables the output functionality of each channel of the digital I/O port.
vtex10xxA_set_dio_pulse	Generates a 1 µs pulse on selected channels of the digital I/O port.
vtex10xxA_set_fifo_config	Sets the data format and overflow behavior of the FIFO memory.
vtex10xxA_set_filt_freq	Sets low-pass filter cutoff frequency of specified channels.
vtex10xxA_set_init_cont	Enables or disables the use of init continuous mode.
vtex10xxA_set_lan_event_domain	Sets the instrument's LAN domain.
vtex10xxA_set_lan_eventlog_enabled	Enables/disables the event log.
vtex10xxA_set_lan_eventlog_overflowmode	Sets the LAN event log overflow mode.
vtex10xxA_set_limit_set0	Sets the limit set 0 values manually for the specified channels.
vtex10xxA_set_limit_set0_manual	Enables or disables manual entry of limit set 0 values for the specified channels.
vtex10xxA_set_limit_set1	Sets the limit set 1 values for the specified channels.
vtex10xxA_set_linear_correction	Sets the mx+b linear correction of a voltage channel.

API	Description
vtex10xxA_set_OTD_enable	Enables/disables open transducer detection for selected channels.
vtex10xxA_set_scanlist	Sets the scan list to be acquired.
vtex10xxA_set_trig_lan_eventID	Sets the event ID of LAN trigger event.
vtex10xxA_set_trig_lan_filter	Sets the filter for the LAN trigger event.
vtex10xxA_set_trig_source_timer	Sets a trigger source of timer only and sets the timer interval.
vtex10xxA_set_trigger_count	Sets the trigger count value.
vtex10xxA_set_trigger_delay	Sets the trigger delay.
vtex10xxA_set_trigger_infinite	Enables or disables the use of an infinite trigger count.
vtex10xxA_set_trigger_source	Sets the trigger source events.
vtex10xxA_set_trigger_sourceEx	Sets the trigger source events including LAN events.
vtex10xxA_set_trigger_timer	Sets the timer interval for a timer source event.
vtex10xxA_set_user_cjc_enable	Enables/ or disables the use of a user-defined CJC temperature for the specified channels.
vtex10xxA_set_user_cjc_temp	Sets the user-defined CJC temperature for the specified channels.
vtex10xxA_set_user_conversion	Sets the user-defined conversion polynomials.
vtex10xxA_set_vtb_output	Sets the static level that each channel of the LXI trigger bus will assume if enabled.
vtex10xxA_set_vtb_output_enable	Enables/ or disables the output functionality of each channel of the LXI trigger bus.
vtex10xxA_set_vtb_pulse	Generates a 1 µs pulse on selected channels of the trigger bus.
vtex10xxA_set_vtb_wiredor	Sets the Wired-OR state for each of the individual LXI Trigger Bus lines.
vtex10xxA_set_vtb_wiredor_bias	Sets the Wired-OR bias state for each LXI Trigger Bus lines.
vtex10xxA_soft_arm	Performs a software arm.
vtex10xxA_soft_trigger	Performs a software trigger.
vtex10xxA_unlock	Releases a lock on the instrument.

SAMPLE FUNCTION DEFINITION

Function_Name

FUNCTION PROTOTYPE

This section provides the exact syntax of the function as it would be written in a program.

FUNCTION PARAMETERS

This section identifies the parameters that are associated with the function. A description of the parameter will be provided and, when appropriate, the range of values that the parameter will accept without creating an error. Ranges are assumed to be inclusive unless otherwise specified.

DATA ITEM RESET VALUE

This section provides the values the data items associated with this function assume after a reset condition. This section is only applicable to “set” functions.

DESCRIPTION

This section details what occurs when this function is called.

EXAMPLE

This section provides an example of how this function might appear in an application.

vtex10xxA_abort

FUNCTION PROTOTYPE

ViStatus vtex10xxA_abort(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function aborts the current acquisition.

EXAMPLE

vtex10xxA_append_scanlist

FUNCTION PROTOTYPE

ViStatus vtex10xxA_append_scanlist (ViSession **vi,** ViInt32 **channels[]**,ViInt32 **numChannels**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels[] = an array of channels that will be examined. Valid input values: 0 through 47.
numChannels = indicates the number of channels in the **channels[]** array.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function adds channels onto the end of an existing scan list.

EXAMPLE

vtex10xxA_break_lock

FUNCTION PROTOTYPE

ViStatus vtex10xxA_break_lock(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function breaks a lock on the instrument. This releases a lock on the instrument, regardless of its owner. This allows for instrument recovery if the locking IP address would become disabled.

NOTE	Breaking a lock on the instrument does not automatically acquire it. That must be done with a separate vtex10xxA_lock call.

EXAMPLE

vtex10xxA_check_lock

FUNCTION PROTOTYPE

ViStatus vtex10xxA_check_lock(ViSession **vi**, **&locked**, **&mine**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
locked = a Boolean value indicating the lock status of the instrument.
mine = a Boolean value indicating whether the lock is owned by the host IP address that issued the query.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the lock status of the instrument. When locked, the EX10xxA will accept commands from only the host IP address that issued the lock command.

EXAMPLE

vtex10xxA_clear_lan_eventlog

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_clear_lan_eventlog(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function clears event log entries in the instrument.

EXAMPLE

vtex10xxA_close

FUNCTION PROTOTYPE

ViStatus vtex10xxA_close (ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function closes an instrument programming session. This function should be performed at the conclusion of the test application. Part of its execution is the unlocking of the instrument, leaving it in the proper state for the next application.

EXAMPLE

vtex10xxA_disable_streaming_data

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_disable_streaming_data(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function disables data streaming.

EXAMPLE

vtex10xxA_enable_streaming_data

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_enable_streaming_data (ViSession **vi**, void ***private_data**, EX10XXA_STREAM_CALLBACK **callback**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
private_data = a pointer to client data that is passed to the callback function.
callback = a call back function to be called when new data is received.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function enables data streaming. Data is returned to the callback function that the user specified. In order to implement this function, the macro INSTR_LANGUAGE_SPECIFIC must be defined.

NOTE	If running the EX10xxA with existing EX1048 code, it is important to note that the time stamp for the EX10xxA has nanosecond resolution (ns), whereas the EX1048 has microsecond resolution (µs). The vtex10xxA_enable_streaming_dataEx call provides for compatibility with the EX1048 via the **legacy_mode** parameter.

The **callback** function syntax is as follows:

ViInt32(*EX10XXA_STREAM_CALLBACK)(void *priv, EX10xxA_SampleData *data);

The **SampleData** structure, which holds each datapage, is as follows:

```
struct EX10xxA_SampleData {
        ViUInt32 ss_secs;               /* data sample time stamp */
        ViUInt32 ss_nsecs;              /* fractional part of time stamp in nanoseconds (microseconds in legacy mode) */
        struct {
            ViUInt32 x_len;             /* array size of data samples */
            ViReal32 *x_val;            /* an array of data samples */
        } x;
        struct {
            ViUInt32 x_counts_len;      /* array size of ADC counts */
            ViInt32 *x_counts_val;      /* an array of 20-bit ADC Readings */
        } x_counts;
        struct {
            ViUInt32 x_ticks_len;       /* array size of x ticks */
            ViUInt16 *x_ticks_val;      /* 16-bit measurement timestamps - 100  ns offsets from scan start time */
        } x_ticks;
        struct {
            ViUInt32 cjc_len;           /* array size of cjc data*/
            ViReal32 *cjc_val;          /* an array of 32-bit floating point CJC values */
        } cjc;
        struct {
            ViUInt32 cjc_counts_len;    /* array size of cjc count */
            ViUInt32 *cjc_counts_val;   /* an array of 20-bit ADC Readings */
        } cjc_counts;
        struct {
            ViUInt32 cjc_ticks_len;     /* array size of cjc ticks */
            ViUInt16 *cjc_ticks_val;    /* 16-bit CJC timestamps - 100  ns offsets from scan start time */
        } cjc_ticks;
        struct {
            ViUInt32 limits_len;        /* array size of limit data */
            ViUInt32 *limits_val;     /* limit data 4 bits for each channel */
```

```
        } limits;
};
```

The data format of the limit values (**limits_val**) is as follows:

index 0: set 0 low limit exceeded, channels 0-31
index 1: set 0 low limit exceeded, channels 32-63
index 2: set 0 high limit exceeded, channels 0-31
index 3: set 0 high limit exceeded, channels 32-63
index 4: set 1 low limit exceeded, channels 0-31
index 5: set 1 low limit exceeded, channels 32-63
index 6: set 1 high limit exceeded, channels 0-31
index 7: set 1 high limit exceeded, channels 32-63

Each bit is "1" if the limit was exceeded and "0" if it was not. Channels 48 through 63 are CJC and special channels which can be ignored by most users.

The ticks value (**ticks_val**) data format is a tightly packed array of unsigned bytes. This is used rather than 16-bit integers as this is the most efficient way for the RPC to package the data. As a result, the length of the data is actually twice the channel count and must be manipulated as follows in order to read the data:

```
uint16_t real_ticks_ch0 = ntohs(datapage.x_ticks.x_ticks_val[0] << 8 | datapage.x_ticks.x_ticks_val[1]);
uint16_t real_ticks_ch1 = ntohs(datapage.x_ticks.x_ticks_val[2] << 8 | datapage.x_ticks.x_ticks_val[3]);
```

or, alternatively:

```
uint16_t real_ticks_ch0 = ntohs(*((uint16_t*)&datapage.x_ticks.x_ticks_val[0]));
uint16_t real_ticks_ch1 = ntohs(*((uint16_t*)&datapage.x_ticks.x_ticks_val[2]));
```

EXAMPLE

See *Retrieving Data (Read FIFO and Streaming Data)* in 0 for an example.

vtex10xxA_enable_streaming_dataEx

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_enable_streaming_dataEx (ViSession **vi**, void ***private_data**, EX10XXA_STREAM_CALLBACK **callback**, ViBoolean **legacy_mode**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
private_data = a pointer to client data that is passed to the callback function.
callback = a call back function to be called when new data is received.
legacy_mode = a Boolean value indicating the time stamp resolution. If set to "1", microsecond (ms) resolution is used. If set to "0", nanosecond (ns) resolution is used.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function enables data streaming. Data is returned to the callback function that the user specified. In order to implement this function, the macro INSTR_LANGUAGE_SPECIFIC must be defined.

NOTE	If running the EX10xxA with existing EX1048 code, it is important to note that the time stamp for the EX10xxA has nanosecond (ns) resolution, whereas the EX1048 has microsecond (µs) resolution. If using the EX1048 Rev 2 *plug&play* driver with the EX1048A, the time stamp will be returned in microseconds.

The **callback** function syntax is as follows:

ViInt32(*EX10XXA_STREAM_CALLBACK)(void *priv, EX10xxA_SampleData *data);

The **callback** data structure is as follows:

```
struct EX10xxA_SampleData {
        ViUInt32 ss_secs;           /* data sample time stamp */
        ViUInt32 ss_nsecs;          /* fractional part of time stamp in nanoseconds (microseconds in legacy mode) */
        struct {
            ViUInt32 x_len;         /* array size of data samples */
            ViReal32 *x_val;        /* an array of data samples */
        } x;
        struct {
            ViUInt32 x_counts_len;      /* array size of ADC counts */
            ViInt32 *x_counts_val;      /* an array of 20-bit ADC Readings */
        } x_counts;
        struct {
            ViUInt32 x_ticks_len;       /* array size of x ticks */
            ViUInt16 *x_ticks_val;      /* 16-bit measurement timestamps - 100  ns offsets from scan start time */
        } x_ticks;
        struct {
            ViUInt32 cjc_len;           /* array size of cjc data*/
            ViReal32 *cjc_val;          /* an array of 32-bit floating point CJC values */
        } cjc;
        struct {
            ViUInt32 cjc_counts_len;    /* array size of cjc count */
            ViUInt32 *cjc_counts_val;   /* an array of 20-bit ADC Readings */
        } cjc_counts;
        struct {
            ViUInt32 cjc_ticks_len;     /* array size of cjc ticks */
            ViUInt16 *cjc_ticks_val;    /* 16-bit CJC timestamps - 100  ns offsets from scan start time */
        } cjc_ticks;
        struct {
```

```
            ViUInt32 limits_len;      /* array size of limit data */
            ViUInt32 *limits_val;     /* limit data 4 bits for each channel */
        } limits;
};
```

The data format of the limit values (**limits_val**) is as follows:

index 0: set 0 low limit exceeded, channels 0-31
index 1: set 0 low limit exceeded, channels 32-63
index 2: set 0 high limit exceeded, channels 0-31
index 3: set 0 high limit exceeded, channels 32-63
index 4: set 1 low limit exceeded, channels 0-31
index 5: set 1 low limit exceeded, channels 32-63
index 6: set 1 high limit exceeded, channels 0-31
index 7: set 1 high limit exceeded, channels 32-63

Each bit is “1” if the limit was exceeded and “0” if it was not. Channels 48 through 63 are CJC and special channels which can be ignored by most users.

The ticks value (**ticks_val**) data format is a tightly packed array of unsigned bytes. This is used rather than 16-bit integers as this is the most efficient way for the RPC to package the data. As a result, the length of the data is actually twice the channel count and must be manipulated as follows in order to read the data:

```
uint16_t real_ticks_ch0 = ntohs(datapage.x_ticks.x_ticks_val[0] << 8 | datapage.x_ticks.x_ticks_val[1]);
uint16_t real_ticks_ch1 = ntohs(datapage.x_ticks.x_ticks_val[2] << 8 | datapage.x_ticks.x_ticks_val[3]);
```

or, alternatively:

```
uint16_t real_ticks_ch0 = ntohs(*((uint16_t*)&datapage.x_ticks.x_ticks_val[0]));
uint16_t real_ticks_ch1 = ntohs(*((uint16_t*)&datapage.x_ticks.x_ticks_val[2]));
```

EXAMPLE

See Retrieving Data (Read FIFO and Streaming Data) in 0 for an example.

vtex10xxA_error_message

FUNCTION PROTOTYPE

ViStatus vtex10xxA_error_message (ViSession **vi**,ViStatus **statusCode**,ViChar _VI_FAR **message[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
statusCode = indicates the status returned by the driver.
message[]=contains the driver error message.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function outputs the error message associated with the **errCode** parameter.

EXAMPLE

vtex10xxA_error_query

FUNCTION PROTOTYPE

ViStatus vtex10xxA_error_query (ViSession **vi**,ViPInt32 **errorCode**,ViChar _VI_FAR **errorMessage[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. The handle is issued by the resource manager and remains valid until the session is closed.
errorCode = points to storage for the error code from the instrument.
errorMessage[] = contains the driver error message.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function outputs the error code and error message obtained from the instrument.

EXAMPLE

vtex10xxA_get_accum_limit_status

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_accum_limit_status(ViSession **vi**, ViBoolean **set0_lower[]**, ViBoolean **set0_upper[]**, ViBoolean **set1_lower[]**, ViBoolean **set1_upper[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
set0_lower[] = a return array of Boolean values representing the accumulated limit status of all 48 channels with respect to each channel's limit set 0 lower limit. Channel 0 through channel 47 are represented in array elements [0] through [47], respectively. If a "1" is returned, the set 0 lower limit has been tripped. If a "0" is returned, the set 0 lower limit has not been tripped.
set0_upper[] = a return array of Boolean values representing the accumulated limit status of all 48 channels with respect to each channel's limit set 0 upper limit. Channel 0 through channel 47 are represented in array elements [0] through [47], respectively. If a "1" is returned, the set 0 upper limit has been tripped. If a "0" is returned, the set 0 upper limit has not been tripped.
set1_lower[] = a return array of Boolean values representing the accumulated limit status of all 48 channels with respect to each channel's limit set 1 lower limit. Channel 0 through channel 47 are represented in array elements [0] through [47], respectively. If a "1" is returned, the set 1 lower limit has been tripped. If a "0" is returned, the set 1 lower limit has not been tripped.
set1_upper[] = a return array of Boolean values representing the accumulated limit status of all 48 channels with respect to each channel's limit set 1 upper limit. Channel 0 through channel 47 are represented in array elements [0] through [47], respectively. If a "1" is returned, the set 1 upper limit has been tripped. If a "0" is returned, the set 1 upper limit has not been tripped.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the accumulated limit status of all 48 channels. Limit status is returned for all channels, regardless of their inclusion in the scan list. The returned values represent, on a per channel basis, any excursion of the measurement data over their respective limits since the last trigger initialize. As implied, limit status is cleared as part of the vtex10xxA_init_imm function.

The limit status for unscanned channels is always 0.

EXAMPLE

vtex10xxA_get_alarm

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_alarm(ViSession **vi**, ViPReal64 **timeSeconds**, ViPReal64 **timeFraction**, ViPReal64 **timePeriod**, ViPInt32 **count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
timeSeconds = a real output value indicating the time (in seconds) when the timer starts. Valid return values: All positive whole numbers.
timeFraction = a real output value indicating the fractional part of time (in seconds) when the timer starts. Valid return values: 0.0 to 0.999999999.
timePeriod = a real output value indicating the time (in seconds) between timer ticks. Valid return values: 0.001 to 4294.0.
count = an integer output value indicating the number of timer ticks. Valid return values: 0.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the LXI alarm, time, interval and count for the alarm source event. The same value is used for both arm and trigger events. Note, the count property is not currently supported and is included for future expansion. It currently always returns “0”, implying that the alarm fires constantly.

EXAMPLE

vtex10xxA_get_alarm_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_alarm_enable (ViSession **vi**, ViPBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
alarm = a Boolean value indicating whether the use of the LXI alarm is enabled. If a “1” is returned, the LXI alarm is enabled. If a “0” is returned, it is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the enabled status of the LXI alarm.

EXAMPLE

vtex10xxA_get_arm_count

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_arm_count(ViSession **vi**, ViPInt32 **count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
count = an integer output value indicating the arm count. Valid return values: 1 through (2^{31}-1).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the arm count value.

EXAMPLE

vtex10xxA_get_arm_delay

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_arm_delay(ViSession **vi**, ViPReal64 **arm_delay**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
arm_delay = a real output value (in seconds) indicating the arm delay. Valid return values: 0 through 4294 (71.5 minutes) with a resolution of 0.000001 (1 µs).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the arm delay, the time between the recognition of the arm event and the transition into the TRIG layer of the trigger model.

EXAMPLE

vtex10xxA_get_arm_infinite

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_arm_infinite(ViSession **vi**, ViPBoolean **arminf**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
arminf = a Boolean value indicating whether the use of an infinite arm count is enabled. If a “1” is returned, the arm count is set to infinite. If a “0” is returned, the arm count is not infinite.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled status of an infinite arm count.

EXAMPLE

vtex10xxA_get_arm_lan_eventID

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_arm_lan_eventID(ViSession **vi**, ViInt32 **lanNumber**, ViInt32 **buffLength**, ViChar **eventID[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = indicates the LAN event line whose arm event will be returned. Valid return values: 0 through 7.
buffLength = a real integer output that sets the length of the character buffer where the **eventID** string is returned.
eventID = the Arm LAN event ID string for the specified LAN event. Valid return values: ASCII strings of length 1 through 16.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns specified event ID of a LAN arm event. The valid value of **lanNumber** is an integer 0 through 7 corresponding to LXI LAN events LAN0 through LAN7.

EXAMPLE

vtex10xxA_get_arm_lan_filter

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_arm_lan_filter(ViSession **vi**, ViInt32 **lanNumber**, ViInt32 **buffLength**, ViChar **filter[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = indicates the LAN event line whose arm event will be returned. Valid return values: 0 through 7.
buffLength = a real integer output in dictating the length of character buffer where the **filter** string is returned.
filter = the Arm LAN filter string for the specified LAN Event. Valid return values: ASCII strings of length 0 to 512 which conform to the filter syntax as defined in the *IVI-3.15 IVILXISync* specification.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the LAN arm event filter for the specified LAN event line. The valid value of **lanNumber** is an integer 0 through 7 corresponding to LXI LAN events LAN0 through LAN7.

EXAMPLE

vtex10xxA_get_arm_source

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_arm_source(**vi**, ViInt16 **vtb_masks[]**, ViInt16 **dio_masks[]**, **ViPBoolean timer_enable**, **ViPBoolean immediate**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks[] = a return array of four 8-bit integer values representing the enabled state of arm events from the 8 channels of the trigger bus. The order of the values is: positive edge, negative edge, positive level, negative level. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid return values: 0 through 255.
dio_masks[] = a return array of four 8-bit integer values representing the enabled state of arm events from the 8 channels of the digital I/O port. The order of the values is: positive edge, negative edge, positive level, negative level. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid return values: 0 through 255.
timer_enable = a Boolean value indicating whether the timer is enabled as an arm event. If a “1” is returned for this parameter, the timer can act as an arm event. If a “0” is returned, the timer is not a valid arm event.
immediate = a Boolean value indicating whether immediate is enabled as an arm event. If a “1” is returned for this parameter, an immediate arm is enabled. If a “0” is returned, an immediate arm is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled arm source events.

EXAMPLE

vtex10xxA_get_arm_sourceEx

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_arm_sourceEx(ViSession **vi**, ViInt16 **vtb_masks[]**,ViInt16 **dio_masks[]**, ViInt16 **lan_masks[]**, ViPBoolean **timer_enable**, ViPBoolean **immediate**, ViPBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
lan_masks = an array of four 8-bit integer values representing the enabling of LAN events from any of the eight LAN events. Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).
timer_enable = a Boolean value indicating whether the timer is enabled as an arm event. If a “1” is returned for this parameter, the timer can act as an arm event. If a “0” is returned, the timer is not a valid arm event.
immediate = a Boolean value indicating whether immediate is enabled as an arm event. If a “1” is returned for this parameter, an immediate arm is enabled. If a “0” is returned, an immediate arm is disabled.
alarm = a Boolean value indicating whether the LXI alarm is used as an ARM event. If a “1” is returned for this parameter, the LXI alarm can act as an arm event. If a “0” is returned, the LXI alarm is not a valid arm event.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled arm source events. For the **vtb_masks**, **dio_masks**, and **lan_masks** parameters, the data items in these arrays are ordered as follows: positive edge, negative edge, positive level, negative level. For the **lan_masks** parameter, however, the positive and negative levels are not supported with LAN events and these bits are ignored. Note that any and all of the parameters above can be enabled simultaneously, with the exception of immediate.

EXAMPLE

vtex10xxA_get_calibration_file

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_calibration_file (ViSession **vi**, ViInt32 **calFileType**,ViInt32 **buffLength**,ViChar _VI_FAR **calibration[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
calFileType = returns the type of calibration that was performed. Valid return values: 0, 1, or 2.
buffLength = indicates the length of the calibration array.
calibration[] = a string buffer where the calibration file data is returned.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the calibration file data associated with the **calFileType** into a string buffer. The valid return values for **calFileType** are:

Decimal Value	#define Symbol	Description
0	VTEX10XXA_CAL_FILE_FACTORY	Factory calibration
1	VTEX10XXA_CAL_FILE_SELF	Self-calibration
2	VTEX10XXA_CAL_FILE_NON_VOL_SELF	Non-volatile self-calibration

EXAMPLE

vtex10xxA_get_calibration_running

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_calibration_running (ViSession **vi**,ViPBoolean **cal_running**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
cal_running = an Boolean return value indicating whether calibration is in progress. If a “1” is returned, this indicates that calibration is in progress. If a “0” is returned, calibration has not been initiated.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

Indicates if calibration is currently in progress.

EXAMPLE

vtex10xxA_get_channel_conversion

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_channel_conversion(ViSession **vi**, ViInt32 **channel**, **ViPInt32 eu_conv**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
eu_conv = an integer output value representing the EU conversion. Valid return values: 0 through 10. See *Description* for more information.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the engineering units (EU) conversion of a specified channel. For the **eu_conv** parameter, the following values may be returned:

Decimal Value	#define Symbol	eu_conv Description
0	VTEX10XXA_CONV_MV	Voltage
1	VTEX10XXA_CONV_THERMO_TYPE_J	Type J
2	VTEX10XXA_CONV_THERMO_TYPE_K	Type K
3	VTEX10XXA_CONV_THERMO_TYPE_T	Type T
4	VTEX10XXA_CONV_THERMO_TYPE_E	Type E
5	VTEX10XXA_CONV_THERMO_TYPE_B	Type B
6	VTEX10XXA_CONV_THERMO_TYPE_S	Type S
7	VTEX10XXA_CONV_THERMO_TYPE_R	Type R
8	VTEX10XXA_CONV_THERMO_TYPE_N	Type N
9	VTEX10XXA_CONV_USER_DEF0	User-defined 0
10	VTEX10XXA_CONV_USER_DEF1	User-defined 1

EXAMPLE

vtex10xxA_get_channel_count

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_channel_count(ViSession **vi**, ViPInt32 **channel_count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
Channel_count = an integer output value representing the number of channels available in the instrument. Valid return values: 0 through 48.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the number of channels available in the instrument.

EXAMPLE

vtex10xxA_get_channel_range

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_channel_range(ViSession **vi**, ViInt32 **channel**, ViPReal64 **range**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the range value is desired. Valid input values: 0 through 47.
range = returns the voltage range value for the specified **channel**. Valid return values: 0.01 (10 mV), 0.067 (±67 mV), 0.1 (100 mV), 1 (1 V), and 10 (10 V).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns channel gain of a voltage channel. An error is returned if a thermocouple channel is specified by the **channel** parameter.

EXAMPLE

vtex10xxA_get_channel_type

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_channel_type(ViSession **vi**, ViInt32 **channel**, ViPInt32 **type**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the range value is desired. Valid input values: 0 through 47.
type = indicates the measurement type of the specified channel. Valid return values: 0 through 3. See *Description* below for more information.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns whether specified channel is voltage or thermocouple channel.

Valid return values for the **type** parameter are:

Decimal Value	#define Symbol	type Description
0	VTEX10XXA_CHANNEL_TYPE_NONE	None
1	VTEX10XXA_CHANNEL_TYPE_TC_ONLY	Thermocouple only
2	VTEX10XXA_CHANNEL_TYPE_GP_VOLT	Voltage only

EXAMPLE

vtex10xxA_get_dio_input

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_input(ViSession **vi**, **ViPInt32 dio_in**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_in = an integer output value in decimal representing the 8-bit value of the port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the current input state of the digital I/O port.

EXAMPLE

```
// check state of DIO bits 7, 4, and 0
        ViInt32 dio_in;
        vtex10xxA_get_dio_input(vi, &dio_in);
        if (dio_in & 0x80)
              printf("Bit 7 is high");
        else printf("Bit 7 is low");
        if (dio_in & 0x10)
              printf("Bit 4 is high");
        else printf("Bit 4 is low");
        if (dio_in & 0x01)
              printf("Bit 0 is high");
        else printf("Bit 0 is low");
```

vtex10xxA_get_dio_limit_event

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_limit_event(ViSession **vi**, ViInt16 **dio_channel**, ViInt16 **limit_masks[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port for which the DIO limit event status is desired. Value must be an integer in the range of 0 to 7.
limit_masks = a return array of 48 4-bit integer values representing, on a per input channel basis, the linking of limit evaluations to any of the four limit conditions. Within the 4-bit field, the order of the values is: limit set 0 lower, limit set 0 upper, limit set 1 lower, limit set 1 upper. Channel 0 through channel 47 are represented in array elements [0] through [47], respectively. Valid return values for the array elements of the **limit masks** parameter: 0 through 15.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled DIO limit events.

EXAMPLE

vtex10xxA_get_dio_limit_event_invert

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_limit_event_invert (ViSession **vi, ViInt16 dio_channel**, ViPBoolean **invert**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port to be queried. Valid input values: range of 0 to 7.
invert = a Boolean value indicating whether the specified DIO channel is operating in invert mode. If a “1” is returned, this indicates that DIO channel operation is inverted. If a “0” is returned, DIO operation is not inverted.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the inverted operation of a DIO channel linked as a limit event.

EXAMPLE

vtex10xxA_get_dio_limit_event_latch

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_limit_event_latch(ViSession **vi**, ViInt16 **dio_channel, ViPBoolean latch**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port to be queried. Value must be an integer in the range of 0 through 7.
latch = a Boolean value indicating whether the specified DIO channel is operating in latch mode. If a “1” is returned for this parameter, indicates that he latch operation has been enabled. If a “0” is returned, the latch operation has not been enabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the latch operation of a DIO channel linked as a limit event.

EXAMPLE

vtex10xxA_get_dio_output

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_output(ViSession **vi**, **ViPInt32 dio_out**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_out = an integer output value in decimal that represents the programmed output state of the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the programmed output state of the digital I/O port. This simply returns the programmed setting. Since the outputs must be enabled, it does not necessarily represent the true output state.

EXAMPLE

```
// query status of DIO bit 4
        ViInt32 dio_out;
        ViInt32 dio_outen;
        vtex10xxA_get_dio_output(vi, &dio_out);
        vtex10xxA_get_dio_output_enable(vi, &dio_outen);
        if (dio_out & 0x10)
              printf("Bit 4 is set high");
        else printf("Bit 4 is set low");
        if (dio_outen & 0x10)
              printf("Bit 4 is enabled");
        else printf("Bit 4 is not enabled");
```

vtex10xxA_get_dio_output_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_dio_output_enable(ViSession **vi**, **ViPInt32 out_enable**)

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
out_enable = an integer output value in decimal that represents the output enable state of the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the output enable state of the digital I/O port.

EXAMPLE

```
// query status of DIO bit 4
        ViInt32 dio_out;
        ViInt32 dio_outen;
        vtex10xxA_get_dio_output(vi, &dio_out);
        vtex10xxA_get_dio_output_enable(vi, &dio_outen);
        if (dio_out & 0x10)
              printf("Bit 4 is set high");
        else printf("Bit 4 is set low");
        if (dio_outen & 0x10)
              printf("Bit 4 is enabled");
        else printf("Bit 4 is not enabled");
```

vtex10xxA_get_fifo_config

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_fifo_config(ViSession **vi**, ViPBoolean **report_cjc**, **ViPBoolean report_timestamp**, **ViPBoolean report_celsius**, **ViPBoolean blocking_mode**)

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
report_cjc = a Boolean value indicating whether CJC temperatures will be reported in the acquisition data. Reported CJC temperatures are always in units of ºC, regardless of the value of the **report_celsius** parameter. If a "1' is returned by this parameter, CJC temperatures will be reported in the acquisition data. If a "0" is returned, CJC values will not be reported.
report_timestamp = a Boolean value indicating whether delta timestamps per channel will be reported in the acquisition data. If a "1' is returned by this parameter, timestamps per channel will be reported in the acquisition data. If a "0" is returned, timestamps per channel will not be reported.
report_celsius = a Boolean value indicating the units the input channel is reporting data in. If a "1" is returned, the units are Celsius (ºC). If a "0" is returned, the units are Fahrenheit (ºF).
blocking_mode = a Boolean value indicating the FIFO behavior upon reaching maximum capacity during scanning. With blocking mode enabled, additional readings are discarded, leaving the contents of the buffer intact. With blocking mode disabled, the buffer becomes circular, in that additional readings overwrite the oldest readings. If a "1" is returned for this parameter, blocking mode is enabled. If a "1" is returned, blocking mode is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the data format and overflow behavior of the FIFO memory.

NOTE	Regardless of the settings of the **report_cjc** and **report_timestamp** parameters, the CJC temperature and delta timestamp information is not accessible through the vtex10xxA_read_fifo function. They are available through the streaming interface and the parameter settings apply for it.

EXAMPLE

vtex10xxA_get_fifo_count

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_fifo_count(ViSession **vi**, **ViPInt32 count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
count = an integer output value indicating the number of data pages. Valid return values: 0 through 187246.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the number of data pages (scans) in the FIFO memory.

EXAMPLE

vtex10xxA_get_filt_freq

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_filt_freq(ViSession **vi**, ViInt32 **channel**,ViPReal64 **filt_freq**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the hardware filter frequency value is desired. Valid input values: 0 through 47.
filt_freq = a real output value representing the frequency setting in hertz (Hz). Valid return values: 4, 15, 40, 100, 500, and 1000.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns current low-pass filter cutoff frequency of a specified channel. The filters on the EX10xxA support cutoff frequencies of 4 Hz, 15 Hz, 40 Hz, 100 Hz, 500 Hz, and 1 kHz.

EXAMPLE

vtex10xxA_get_init_cont

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_init_cont(ViSession **vi**, ViPBoolean **init_cont_mode**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
init_cont_mode = a Boolean value indicating whether init continuous mode is enabled. If a “1” is returned, init continuous mode is enabled. If a “0” is returned, init continuous mode is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled status of init continuous mode.

EXAMPLE

vtex10xxA_get_lan_event_domain

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_lan_event_domain (ViSession **vi**, ViPInt32 **domain**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
domain = returns the LAN domain for the EX10xxA. Valid return value: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the LAN domain of the EX10xxA.

EXAMPLE

vtex10xxA_get_lan_event_source_state

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_lan_event_source_state(ViSession **vi**,ViPInt32 **arm_state**,ViPInt32 **trig_state**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
arm_state = returns an integer output value representing the 8-bit value of the port. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid return values are: 0 to 255.
trig_state = returns an integer output value representing the 8-bit value of the port. Within the 8-bit filed, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid return values are: 0 to 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the current arm and trigger state of the LAN events.

EXAMPLE

vtex10xxA_get_lan_eventlog_count

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_get_lan_eventlog_count(ViSession **vi**, ViPInt32 **logCount**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
logCount = indicates the number of event log entries.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the number of event log entries made by the EX10xxA See vtex10xxA_set_lan_eventlog_enabled for more details.

EXAMPLE

vtex10xxA_get_lan_eventlog_enabled

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_get_lan_eventlog_enabled(ViSession **vi**, ViPBoolean **enable**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
enable = a Boolean value indicating whether LAN event logging is enabled. If a “1” is returned, logging is enabled. If a “0” is returned, logging is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the status of the event log.

EXAMPLE

vtex10xxA_get_lan_eventlog_overflowmode

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_lan_eventlog_overflowmode (ViSession **vi**,ViPInt32 **mode**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
mode = indicates whether the oldest event log shall be overwritten (VTEX10XXA_EVENTLOG_OVERFLOWMODE_OVERWRITE) or the new event log shall be ignored (VTEX10XXA_EVENTLOG_OVERFLOWMODE_IGNORE) when the event log buffer is full.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the LAN event log overflow mode. In the event that the LAN event log buffer becomes full, the EX10xxA will either overwrite the existing log or will not write the new log information.

The **mode** parameter has the following valid return values:

Decimal Value	#define Symbol	mode Description
0	VTEX10XXA_EVENTLOG_OVERFLOWMODE_OVERWRITE	Overwrites oldest event log buffer.
1	VTEX10XXA_EVENTLOG_OVERFLOWMODE_IGNORE	Ignores new event log data.

EXAMPLE

vtex10xxA_get_limit_set0

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_limit_set0(ViSession **vi**, ViInt32 **channel**, **ViPReal64 lower_limit**, **ViPReal64 upper_limit**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the limit set 0 values are desired. Valid input values: 0 through 47.
lower_limit = a real output value indicating the lower limit. Valid return values: -3e+38 through 3e+38.
upper_limit = a real output value indicating the upper limit. Valid return values: -3e+38 through 3e+38.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the limit set 0 values of a specified channel.

EXAMPLE

vtex10xxA_get_limit_set0_manual

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_limit_set0_manual(ViSession **vi**, ViInt32 **channel**, **ViPBoolean manual**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the manual entry control value is desired. Valid input values: 0 to 47.
manual = a Boolean value indicating whether manual entry is enabled. If a “1” is returned, manual entry is enabled. If a “0” is returned, manual entry is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the manual entry control of limit set 0 values of a specified channel.

EXAMPLE

vtex10xxA_get_limit_set1

FUNCTION PROTOTYPE

vtex1048_get_limit_set1(ViSession **vi**, ViInt32 **channel**, **ViPReal64 lower_limit, ViPReal64 upper_limit**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the limit set 0 values are desired. Valid input values: 0 through 47.
lower_limit = a real output value indicating the lower limit. Valid return values: -3e+38 through 3e+38.
upper_limit = a real output value indicating the upper limit. Valid return values: -3e+38 through 3e+38.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the limit set 1 values of a specified channel.

EXAMPLE

vtex10xxA_get_linear_correction

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_get_linear_correction(ViSession **vi**, ViInt32 **channel**,ViPReal64 **gain**, ViPReal64 **offset**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the limit set 0 values are desired. Valid input values: 0 through 47.
gain = A linear correction gain. ‘m’ of m(x-b) conversion.
offset = A linear correction offset. ‘b’ of m(x-b) conversion.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns linear correction of a voltage channel. If the channel specified by the **channel** parameter is a thermocouple channel, an error is returned.

EXAMPLE

vtex10xxA_get_model

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_get_model(ViSession **vi**, ViInt32 **buffLength**, ViChar[] **model**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
buffLength = indicates the size of the character buffer being passed to the function.
model = indicates the instrument model name.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the instrument's model name.

EXAMPLE

vtex10xxA_get_OTD_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_OTD_enable(ViSession **vi**, ViInt32 **channel**,ViPBoolean **otd**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = indicates the channel for which the enabled status is desired. Valid return values: 0 to 47.
otd = a Boolean value indicating whether the use of a user-defined CJC temperature is enabled. A return value of "0" indicates that user-defined CJC temperature is disabled, while a value of "1" indicates that it is enabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the enabled status of an OTD (open transducer detection) of a specified channel.

EXAMPLE

vtex10xxA_get_ptp_info

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_ptp_info (ViSession **vi**, ViPBoolean **isMaster**,ViPBoolean **isSynchronized**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
isMaster = a Boolean return value that indicates whether or not the device is a master with respect to the precision time protocol and IEEE 1588.
isSynchronized = a Boolean return value that indicates whether or not the device is synchronized with respect to the precision time protocol and IEEE 1588.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns information associated with the precision time protocol implemented on the EX10xxA. Specifically, it returns whether or not the device is a master with respect to PTP and returns whether the device is synchronized.

EXAMPLE

vtex10xxA_get_scanlist

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_scanlist(ViSession **vi**, ViInt32 **channels[]**, **ViPInt32 numChannels**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid return values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid return values: 1 through 48.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the current scan list.

EXAMPLE

vtex10xxA_get_serialNumber

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_serialNumber(ViSession **vi**, ViInt32 **serial**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
serial = an integer output value that indicates the instrument's serial number. Valid return value: 0 through 999999.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns instrument's serial number.

EXAMPLE

vtex10xxA_get_shield_gnd_config

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_shield_gnd_config (ViSession vi, ViPInt32 shield_gnd);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
shield_gnd = an integer output indicating to receive shield ground information from the instrument. Valid return values: 0 through 2

VTEX10XXA_SHLD_GND_NONE = 0
VTEX10XXA_SHLD_GND_SIGNAL = 1
VTEX10XXA_SHLD_GND_CHASSIS = 2

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function is to get whether the shield connected to Signal Ground, Chassis or left disconnected. This is only supported for RX10XX series instruments.

EXAMPLE

vtex10xxA_get_system_time

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_system_time (ViSession **vi**, ViPReal64 **seconds**,ViPReal64 **fractional_seconds**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
seconds = a real return value that indicates the current value of the system time in seconds. This value should be used in combination with the **fractional_seconds** return value to obtain the complete system time.
fractional_seconds = a real return value that indicates the current value of the system time in fractional seconds. The value returned is the fractional time in nanoseconds. This value should be used in combination with the **seconds** return value to obtain the complete system time.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the value for the current system time in seconds.

EXAMPLE

vtex10xxA_get_trig_lan_eventID

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_trig_lan_eventID(ViSession **vi**, ViInt32 **lanNumber**, ViInt32 **buffLength**, ViChar **eventID[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = indicates the LAN event line whose arm event will be returned. Valid return values: 0 through 7.
buffLength = a real integer output that dictates the length of character buffer where the **event ID** parameter is returned.
eventID = the Trigger LAN event ID string for the specified LAN Event. Valid return values: ASCII strings of length 1 through 16.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns specified event ID of a LAN trigger event. The valid value of **lanNumber** is an integer number 0 through 7 corresponding to LXI LAN0 through LAN7 events.

EXAMPLE

vtex10xxA_get_trig_lan_filter

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_trig_lan_filter(ViSession **vi**, ViInt32 **lanNumber**, ViInt32 **buffLength**, ViChar **filter[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = indicates the LAN event line whose arm event will be returned. Valid return values: 0 through 7.
buffLength = a real integer output in dictating the length of character buffer where the **filter** string is returned.
filter = the Trigger LAN filter string for the specified LAN Event. Valid return values: ASCII strings of length 0 through 512 which conform to the filter syntax defined in *IVI-3.15 IVILXISync* specification.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns specified filter of a LAN trigger event. The valid value of **lanNumber** is an integer number 0 through 7 corresponding to LXI LAN0 through LAN7 events.

EXAMPLE

vtex10xxA_get_trigger_count

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_trigger_count(ViSession **vi**, **ViPInt32 count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
count = an integer output value indicating the trigger count. Valid return value: 1 through (2^{31}-1).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the trigger count value.

EXAMPLE

vtex10xxA_get_trigger_delay

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_trigger_delay(ViSession **vi**, **ViPReal64 trig_delay**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
trig_delay = a real output value (in seconds) indicating the trigger delay. Valid return values: 0 through 4294 (71.5 minutes) with a resolution of 0.000001 (1 µs).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the trigger delay, the time between the recognition of the trigger event and the execution of the scan list.

EXAMPLE

vtex10xxA_get_trigger_infinite

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_trigger_infinite(ViSession **vi**, **ViPBoolean triginf**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
triginf = a Boolean value indicating whether the use of an infinite trigger count is enabled. If a “1” is returned, infinite trigger count is enabled. If a “0” is returned, this function is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled status of an infinite trigger count.

EXAMPLE

vtex10xxA_get_trigger_source

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_trigger_source (ViSession **vi**,ViInt16 _VI_FAR **vtb_masks[]**,ViInt16 _VI_FAR **dio_masks[]**,ViPBoolean **timer_enable**,ViPBoolean **immediate**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = A return array of four 8-bit integer values representing the enabled state of trigger events from the 8 channels of the trigger bus. The order of the values is: Positive edge, negative edge, positive level, negative level. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = A return array of four 8-bit integer values representing the enabled state of trigger events from the 8 channels of the digital I/O port. The order of the values is: Positive edge, negative edge, positive level, negative level. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
timer_enable = A Boolean value indicating whether the timer is enabled as a trigger event. If a "0" is returned for this parameter, timer triggering is disabled.
immediate = A Boolean value indicating whether immediate is enabled as a trigger event. If a "0" is returned for this parameter, immediate triggering is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled trigger source events. Regardless of the response, software triggers are always enabled. For the **vtb_masks**, **dio_masks**, and **lan_masks** parameters, the data items in these arrays are ordered as follows: positive edge, negative edge, positive level, negative level. For the **lan_masks** parameter, however, the positive and negative level are not supported with LAN events and these bits are ignored.

NOTE	This function is included for backward compatibility with the EX1048. In order to use this function, the EX10xxA compatibility driver must be used.

EXAMPLE

vtex10xxA_get_trigger_sourceEx

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_get_trigger_sourceEx(ViSession **vi**, ViInt16 **vtb_masks[]**, ViInt16 **dio_masks[]**, ViInt16 **lan_masks[]**, ViPBoolean **timer**, ViPBoolean **immediate,** ViBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
lan_masks = an array of four 8-bit integer values representing the enabling of LAN events from any of the eight LAN events. Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0. Valid input values: 0 (0x00h) through 255 (0xFFh).
timer_enable = a Boolean value indicating whether the timer is enabled as a trigger event. If a "1" is returned for this parameter, the timer is enabled as a trigger. If a "0" is returned for this parameter, timer triggering is disabled.
immediate = a Boolean value indicating whether immediate is enabled as a trigger event. If a "1" is returned for this parameter, immediate triggering is enabled. If a "0" is returned for this parameter, immediate triggering is disabled.
alarm = a Boolean value indicating whether the LXI alarm is used as an ARM event. If a "1" is returned for this parameter, the LXI alarm can act as an arm event. If a "0" is returned, the LXI alarm is not a valid arm event.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled trigger source events. Regardless of the response, software triggers are always enabled. For the **vtb_masks**, **dio_masks**, and **lan_masks** parameters, the data items in these arrays are ordered as follows: positive edge, negative edge, positive level, negative level. For the **lan_masks** parameter, however, the positive and negative level are not supported with LAN events and these bits are ignored.

NOTE	This function supersedes the vtex1048_get_trigger_source () function found in the EX1048A.

EXAMPLE

vtex10xxA_get_trigger_timer

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_trigger_timer (ViSession **vi**, ViPReal64 **trig_timer**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
trig_timer = a real output value indicating the time (in seconds) between timer ticks. Valid return values: 0.001 to 4294.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries the timer interval for the timer source event. The same value is used for both arm and trigger events.

EXAMPLE

vtex10xxA_get_user_cjc_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_user_cjc_enable(ViSession **vi**, ViInt32 **channel**, **ViPBoolean usercjc**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the enabled status is desired. Valid input values: 0 through 47.
usercjc = a Boolean value indicating whether the use of a user-defined CJC temperature is enabled. If a “1” is returned, user-defined CJC temperature is enabled. If a “0” is returned, user-defined CJC temperature is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the enabled status of a user-defined CJC temperature of a specified channel.

EXAMPLE

vtex10xxA_get_user_cjc_temp

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_user_cjc_temp(ViSession **vi**, ViInt32 **channel**, **ViPReal64 cjc_temp**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channel = the channel for which the user-defined CJC temperature is desired. Valid input values: 0 through 47.
cjc_temp = a real output value indicating the CJC temperature in ºC. Valid return values: -3e+38 through 3e+38.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the user-defined CJC temperature of a specified channel.

EXAMPLE

vtex10xxA_get_user_conversion

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_user_conversion(ViSession **vi**, **eu_conv**, **fwdcoeff[]**, **invcoeff[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
eu_conv = the polynomial set to be obtained. Value must be an integer equal to 9 (User0) or 10 (User1).
fwdcoeff[] = a return array of forward conversion polynomial coefficients. Coefficients c_0 through c_{12} are represented in array elements [0] through [12], respectively. Valid return values: -3e+38 to 3e+38.
invcoeff[] = a return array of inverse conversion polynomial coefficients. Coefficients d_0 through d_{12} are represented in array elements [0] through [12], respectively. Valid return values: -3e+38 to 3e+38.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the user-defined conversion polynomials. All twelve elements of the arrays are returned, regardless of how many were set with the vtex10xxA_set_user_conversion function.

The valid return values for the **eu_conv** parameter are:

Decimal Value	#define Symbol	eu_conv Description
0	VTEX10XXA_CONV_MV	Voltage
1	VTEX10XXA_CONV_THERMO_TYPE_J	Type J
2	VTEX10XXA_CONV_THERMO_TYPE_K	Type K
3	VTEX10XXA_CONV_THERMO_TYPE_T	Type T
4	VTEX10XXA_CONV_THERMO_TYPE_E	Type E
5	VTEX10XXA_CONV_THERMO_TYPE_B	Type B
6	VTEX10XXA_CONV_THERMO_TYPE_S	Type S
7	VTEX10XXA_CONV_THERMO_TYPE_R	Type R
8	VTEX10XXA_CONV_THERMO_TYPE_N	Type N
9	VTEX10XXA_CONV_USER_DEF0	User-defined 0
10	VTEX10XXA_CONV_USER_DEF1	User-defined 1

EXAMPLE

vtex10xxA_get_vtb_input

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_vtb_input(ViSession **vi**, **ViPInt32 vtb_in**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_in = an integer output value in decimal representing the 8-bit value of the port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the current input state of the trigger bus.

EXAMPLE

```
// check state of VTB bits 7, 4, and 0
    ViInt32 vtb_in;
    vtex10xxA_get_vtb_input(vi, &vtb_in);
    if (vtb_in & 0x80)
       printf("Bit 7 is high");
    else printf("Bit 7 is low");
    if (vtb_in & 0x10)
       printf("Bit 4 is high");
    else printf("Bit 4 is low");
    if (vtb_in & 0x01)
       printf("Bit 0 is high");
    else printf("Bit 0 is low");
```

vtex10xxA_get_vtb_output

FUNCTION PROTOTYPE

vtex1048_get_vtb_output(ViSession **vi**, **ViPInt32 vtb_out**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_out = an integer output value in decimal that represents the programmed output state of the 8-bit port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the programmed output state of the trigger bus. This simply returns the programmed setting. Since the outputs must be enabled, it does not necessarily represent the true output state.

EXAMPLE

```
// query status of VTB bit 4
        ViInt32 vtb_out;
        ViInt32 vtb_outen;
        vtex10xxA_get_vtb_output(vi, &vtb_out);
        vtex10xxA_get_vtb_output_enable(vi, &vtb_outen);
        if (vtb_out & 0x10)
              printf("Bit 4 is set high");
        else printf("Bit 4 is set low");
        if (vtb_outen & 0x10)
              printf("Bit 4 is enabled");
        else printf("Bit 4 is not enabled");
```

vtex10xxA_get_vtb_output_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_vtb_output_enable(ViSession **vi**, **ViPInt32 out_enable**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
out_enable = an integer output value in decimal that represents the output enable state of the 8-bit port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid return values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the output enable state of the trigger bus.

EXAMPLE

```
// query status of VTB bit 4
        ViInt32 vtb_out;
        ViInt32 vtb_outen;
        vtex10xxA_get_vtb_output(vi, &vtb_out);
        vtex10xxA_get_vtb_output_enable(vi, &vtb_outen);
        if (vtb_out & 0x10)
            printf("Bit 4 is set high");
        else printf("Bit 4 is set low");
        if (vtb_outen & 0x10)
            printf("Bit 4 is enabled");
        else printf("Bit 4 is not enabled");
```

vtex10xxA_get_vtb_wiredor

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_vtb_wiredor(ViSession **vi**, ViPInt32 **wiredor**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
wiredor = an integer return value that indicates the Wired-OR state for all eight LXI Trigger Bus lines. Only the least significant bits are used. The least significant bit of the integer represents LXI trigger bus line 0, where the 8th least significant bit corresponds to LXI trigger bus line 7. A return value of "1" for any of the eight bits indicates that the wired or mode is enabled for that particular LXI trigger bus line, while a value of "0" indicates that Wired-OR is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the Wired-OR state for each individual LXI Trigger Bus channel.

EXAMPLE

vtex10xxA_get_vtb_wiredor_bias

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_vtb_wiredor(ViSession **vi**, ViPInt32 **wiredor**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
wiredor = an integer return value that indicates the Wired-OR state for all eight LXI Trigger Bus lines. Only the least significant bits are used. The least significant bit of the integer represents LXI trigger bus line 0, where the 8th least significant bit corresponds to LXI trigger bus line 7. A return value of "1" for any of the eight bits indicates that the wired or mode is enabled for that particular LXI trigger bus line, while a value of "0" indicates that Wired-OR is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function queries and returns the Wired-OR bias enabled state for each individual LXI trigger bus channel.

EXAMPLE

vtex10xxA_init

FUNCTION PROTOTYPE

ViStatus vtex10xxA_init(ViRsrc **resourceName**, ViBoolean **IDQuery**, ViBoolean **resetDevice**,ViPSession **vi**);

FUNCTION PARAMETERS

resourceName = the VISA resource string. It has the form of "TCPIP::<W.X.Y.Z>::INSTR", where W.X.Y.Z represents the IP address to which to connect.
IDQuery = a Boolean value indicating whether to perform confirmation that the connected instrument is an EX10xxA.
resetDevice = a Boolean value indicating whether to reset the instrument upon connecting.
vi = the session handle (ID), unique to each connection instance.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function opens an instrument programming session. This function must be successfully performed in order to communicate with the EX10xxA. Sessions to multiple instruments can be opened within the same application, each uniquely identified by their session handle.

EXAMPLE

```
#include <vtex10xxA.h>

#define     INSTR_RESRC_STR          "TCPIP::192.168.0.127::INSTR"

ViSession vi;

// open a session to EX10xxA at IP address 192.168.0.127
vtex10xxA_init(INSTR_RESRC_STR, 1, 1, &vi);
```

vtex10xxA_init_imm

FUNCTION PROTOTYPE

ViStatus vtex10xxA_init_imm(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs a trigger initialize, transitioning the trigger model out of the IDLE layer.

EXAMPLE

vtex10xxA_lock

FUNCTION PROTOTYPE

ViStatus vtex10xxA_lock(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function attempts to acquire a lock on the instrument. When locked, the EX10xxA will accept function calls from only the host IP address that issued the lock command. A lock can only be acquired if the instrument is not already locked by another user.

By design, the locking mechanism is able to be overridden by a secondary host that issues a vtex10xxA_break_lock function. Thus, the lock provides a warning to other users that the unit is in a protected operation state, but not absolute security.

The lock status of the instrument is unaffected by the vtex10xxA_reset function.

Self-calibration requires the acquisition of a lock prior to its initiation.

EXAMPLE

vtex10xxA_pop_logged_LAN_event

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_pop_logged_LAN_event(ViSession **vi**, ViInt32 **buffLength**, ViChar **poppedEvent[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
buffLength = an integer return value indicating the length of character buffer where event log string is returned. This parameter should not exceed 256 characters in length.
poppedEvent = the buffer to return the event log entry into. Valid input/return values: ASCII string of length 0 to **buffLength**.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the oldest event log entry in the instrument. The entry will be cleared once it is retrieved. For more information, see the vtex10xxA_get_lan_eventlog_enabled function.

EXAMPLE

vtex10xxA_read_fifo

FUNCTION PROTOTYPE

ViStatus vtex10xxA_read_fifo(ViSession **vi**, ViInt32 **maxscans**, ViReal64 **ts_secs[]**, ViReal64 **ts_fsecs[]**, **ViPInt32 numscans**, ViInt32 **maxdata**, ViReal64 **data[]**, **ViPInt32 numdata**, ViInt32 **to_secs**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
maxscans = the maximum number of scans to return. Valid return values: 1 through (2^{31}-1).
ts_secs[] = a return array of scan start times, specified in seconds since the epoch (Jan. 1, 1970).
ts_fsecs[] = a return array of scan start times, specified in seconds since the last full second represented in **ts_secs[]**.
numscans = an integer output value indicating the actual number of scans retrieved. Valid
maxdata = the maximum length of the return data array. Valid return values: 1 through (2^{31}-1).
data[] = a return array of sample data.
numdata = an integer output value indicating the actual number of samples retrieved. Valid return values: 1 through (2^{31}-1).
to_secs = the timeout period (in seconds), indicating how long to poll the EX10xxA for data. Valid return values: 0 through (2^{31}-1), where 0 represents an infinite timeout period.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function retrieves acquisition data. Data returned by this function includes input channel measurement data and the start time of each scan.

NOTE	In order to provide the maximum reading buffer capacity for future acquisitions, data is deleted from the FIFO memory upon retrieval.

EXAMPLE

```
ViInt32   channels[5] = {0, 1, 2, 3, 4};
ViReal64  ts_secs[20], ts_fsecs[20], data[100];
ViInt32   num_scans, num_data;

vtex10xxA_set_scanlist(vi, channels, 5);
vtex10xxA_set_trig_source_timer(vi, 0.01);
vtex10xxA_set_trigger_count(vi, 20);
vtex10xxA_init_imm(vi);
vtex10xxA_read_fifo(vi, 20, ts_secs, ts_fsecs, &num_scans, 100, data, &num_data, 3);
```

vtex10xxA_read_fifoEx

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_read_fifoEx(ViSession **vi**, ViInt32 **maxscans**, ViReal64 **ts_secs[]**, ViReal64 **ts_fsecs[]**, ViPInt32 **numscans**, ViInt32 **maxdata**, ViReal64 **data[]**, ViPInt32 **numdata**, ViInt32 **maxcjcdata**, ViReal64 **cjcdata[]**, ViPInt32 **numcjcdata**, ViInt32 **to_secs**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
maxscans = the maximum number of scans to return. Valid return values: 1 through (2^{31}-1).
ts_secs[] = a return array of scan start times, specified in seconds since the epoch (Jan. 1, 1970).
ts_fsecs[] = a return array of scan start times, specified in seconds since the last full second represented in **ts_secs[]**.
numscans = an integer output value indicating the actual number of scans retrieved. Valid
maxdata = the maximum length of the return data array. Valid return values: 1 through (2^{31}-1).
data[] = a return array of sample data.
numdata = an integer output value indicating the actual number of samples retrieved. Valid return values: 1 through (2^{31}-1).
maxcjcdata = the maximum length of the return CJC data array. Valid return values: 1 through (2^{31}-1).
cjcdata = a return array of CJC data.
numcjcdata = an integer output value indicating the actual number of CJC data retrieved. Valid return values: 1 through (2^{31}-1).
to_secs = the timeout period (in seconds), indicating how long to poll the EX10xxA for data. Valid return values: 0 through (2^{31}-1), where 0 represents an infinite timeout period.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function retrieves acquisition data with timestamps and CJC data. Data returned by this function includes input channel measurement data and the start time of each scan.

NOTE	In order to provide the maximum reading buffer capacity for future acquisitions, data is deleted from the FIFO memory upon retrieval.

EXAMPLE

```
ViInt32   channels[5] = {0, 1, 2, 3, 4};
ViReal64  ts_secs[20], ts_fsecs[20], data[100];
```

vtex10xxA_reset

FUNCTION PROTOTYPE

ViStatus vtex10xxA_reset(ViSession **vi);**

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs an instrument reset, returning all of the EX10xxA's acquisition configuration parameters to their default values.

NOTE	An instrument reset clears the FIFO reading memory. All desired acquisition data must be retrieved from the FIFO prior to the issuance of this function.

EXAMPLE

vtex10xxA_reset_fifo

FUNCTION PROTOTYPE

ViStatus vtex10xxA_reset_fifo(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function clears the FIFO memory. Since the memory is cleared upon the receipt of a trigger initialize command and made available as data is retrieved, this command is normally not needed. That is, the FIFO need not be specifically cleared before a new acquisition is initiated.

EXAMPLE

vtex10xxA_reset_trigger_arm

FUNCTION PROTOTYPE

vtex1048_reset_trigger_arm(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs a reset of the trigger configuration parameters to default values, while not affecting any other acquisition parameters. In contrast, the vtex10xxA_reset function resets all of the acquisition parameters to their default values.

EXAMPLE

vtex10xxA_revision_query

FUNCTION PROTOTYPE

ViStatus vtex10xxA_revisionQuery(ViSession **vi**, ViChar **driverRev[]**, ViChar **instrRev[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
driverRev[] = a return array representing the release revision of the instrument driver.
instrRev[] = a return array representing the release revision of the embedded firmware.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the release revision of the instrument driver and embedded firmware.

EXAMPLE

```
ViChar      driverRev[256], instrRev[256];

vtex10xxA_revisionQuery(vi, driverRev, instrRev);
```

vtex10xxA_self_cal_clear

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_clear(ViSession vi);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function clears the current self cal data. This operation clears the volatile data, but does not affect any self-calibration data that is stored in nonvolatile memory.

EXAMPLE

vtex10xxA_self_cal_clear_stored

FUNCTION PROTOTYPE

vtex1048_self_cal_clear_stored(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function clears self-calibration data from nonvolatile memory. This operation does not clear the current self-calibration data, only that in non-volatile memory. If this function is called and no non-volatile self-calibration data is present, an error is generated.

EXAMPLE

vtex10xxA_self_cal_get_status

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_get_status(ViSession **vi, ViPInt32 cal_percent**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
cal_percent = an integer output value in decimal that represents the percentage completion status of self-calibration. Valid return values: 0 through 100.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the completion status of self-calibration.

NOTE	Additional instrument driver calls should not be performed until the completion status reaches 100 percent.

EXAMPLE

vtex10xxA_self_cal_init

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_init(ViSession **vi, ViPInt32 override**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
override = an integer output value in decimal that represents an override code. If self-calibration is attempted before the EX10xxA has been powered on continuously for 60 minutes, an error will be generated, and an integer value will be placed in the **override** variable. If appropriate, resending the function will override the error and initiate self-calibration.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs an instrument self-calibration. In general, self-calibration should not be performed until the EX10xxA has been powered on continuously for 60 minutes. In fact, if self-calibration is attempted prior to the required uptime, an error will be generated. When appropriate, as described below, this error can be overridden by resending the function.

NOTES

1) In order to perform a self-calibration, a lock on the instrument must first be acquired. Attempting to self-calibrate without the acquisition of a lock will generate an error that is not able to be overridden. See the vtex10xxA_lock function.

2) The self-calibration uptime requirement is in place to protect the measurement integrity of the instrument. Overriding the requirement must only be done when the operating conditions allow it. An example of this is where the unit has actually been warmed up, but has simply been subjected to a quick power cycle or reboot. **In order to ensure that the override is intentional, it is strongly recommended that user intervention be required in the software application to employ it.**

3) Once self-calibration has been successfully initiated, its percentage completion status is accessible through the vtex10xxA_self_cal_get_status function. **Additional instrument driver calls should not be performed until the completion status reaches 100 percent.** To determine if calibration is still running, use the **vtex10xxA_get_calibration_running** function. If calibration fails, it status will not reach 100%.

EXAMPLE

vtex10xxA_self_cal_is_stored

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_is_stored(ViSession **vi, ViPBoolean stored**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
stored = a Boolean value indicating whether self-calibration data is stored in non-volatile memory. If a “1” is returned for this parameter, this indicates that self-calibration data is stored in non-volatile memory. If a “0” is returned, no self-calibration data is stored.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function returns the presence of self-calibration data in non-volatile memory. Non-volatile self-calibration data is automatically loaded and used upon an instrument power cycle or reset. It is stored with the vtex10xxA_self_cal_store function.

EXAMPLE

vtex10xxA_self_cal_load

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_load(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function loads nonvolatile self-calibration data as the current self-calibration data. If current self-calibration data previously existed, it is simply overwritten and need not be cleared in advance. If this function is called when no non-volatile self-calibration data is present, an error is generated.

EXAMPLE

vtex10xxA_self_cal_store

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_store(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function stores the current self-calibration data into non-volatile memory, enabling it to be loaded upon instrument power cycle and reset. If this function is called when no current self-calibration data is present, an error is generated. Since the existence of non-volatile self-calibration data represents a permanent (although revocable) change from the factory calibration settings, its presence is able to be queried. For more information, see the vtex10xxA_self_cal_is_stored function.

EXAMPLE

vtex10xxA_self_test

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_test (ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function is currently supported only in RX10xx series Calling this function will perform self test operation in **synchronous** mode.

EXAMPLE

vtex10xxA_self_test_init

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_test_init (ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function is currently supported only in RX10xx series Calling this function will perform self test operation in **asynchronous** mode.

EXAMPLE

vtex10xxA_set_alarm

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_alarm(ViSession **vi**, ViReal64 **timeSeconds**, ViReal64 **timeFraction**, ViReal64 **timePeriod**, ViInt32 **count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
timeSeconds = a real input value indicating the whole part of the IEEE1588 time (in seconds) at which the timer starts. Valid input values: Whole numbers 0 through 4294967296.
timeFraction = a real input value indicating the fractional part of the IEEE1588 time (in seconds) at which the timer starts. Valid return values: 0.0 through 0.999999999.
timePeriod = a real input value indicating the time (in seconds) between timer ticks. Valid input values: 0.001 to 4294.0.
count = an integer input value indicating the number of timer ticks. Note that a value of "0" indicates forever. Valid input values: 0.

DATA ITEM RESET VALUE

timeSeconds = 0.0
timeFraction = 0.0
timePeriod = 0.1
count = 0

DESCRIPTION

This function sets the alarm time, period, and count for the LXI alarm source event. The same value is used for both arm and trigger events. The **count** property is currently unsupported but is provided in this API for future expansion. Currently, the only valid value is "0".

EXAMPLE

vtex10xxA_set_alarm_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_alarm_enable (ViSession **vi**,ViBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
alarm = a Boolean value indicating whether an infinite trigger timer is enabled (1) or disabled (0).

DATA ITEM RESET VALUE

alarm = 0

DESCRIPTION

This function enables or disables the use of the LXI alarm. When enabled, the LXI alarm will generate a trigger or arm event.

vtex10xxA_self_test_init

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_test_init (ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function is currently supported only in RX10xx series Calling this function will perform self test operation in **asynchronous** mode.

EXAMPLE

vtex10xxA_set_alarm function.

EXAMPLE

vtex10xxA_set_arm_count

FUNCTION PROTOTYPE

vtex1048_set_arm_count(ViSession **vi**, ViInt32 **count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
count =sets the arm count value. Valid input values: 1 through (2^{31}-1).

DATA ITEM RESET VALUE

count = 1

DESCRIPTION

This function sets the arm count value. This value is reset with each trigger initialize or automatically upon reaching zero when init continuous is enabled.

EXAMPLE

```
// set an arm count of 10
vtex10xxA_set_arm_count(vi, 10);
```

vtex10xxA_set_arm_delay

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_arm_delay(ViSession **vi**, ViReal64 **arm_delay**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
arm_delay = sets the delay value (in seconds). Valid input values: 0 through 4294 (71.5 minutes) with a resolution of 0.000001 (1 µs).

DATA ITEM RESET VALUE

arm_delay = 0.

DESCRIPTION

This function sets the arm delay, the time between the recognition of the arm event and the transition into the TRIG layer of the trigger model.

EXAMPLE

```
// set an arm delay of 5 ms
vtex10xxA_set_arm_delay(vi, 0.005);
```

vtex10xxA_set_arm_infinite

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_arm_infinite(ViSessoin **vi**, ViBoolean **arminf**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
arminf = a Boolean value indicating whether to set the arm count to infinite. If set to “1”, the arm count is set to infinity. If set to “0”, the arm count is not infinite.

DATA ITEM RESET VALUE

arminf = 0.

DESCRIPTION

This function enables or disables the use of an infinite arm count. Enabling overrides any manual setting of arm count.

EXAMPLE

vtex10xxA_set_arm_lan_eventID

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_arm_lan_eventID(ViSession **vi**, ViInt32 **lanNumber**, ViChar **eventID[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = an integer input value indicating the LAN event number corresponding to LAN0 through LAN7. Valid input values: 0 through 7.
eventID = the Arm LAN event ID string for the specified LAN Event. Valid input values: ASCII strings of length 1 through 16.

DATA ITEM RESET VALUE

lanNumber = 0, **eventID** = LAN0
lanNumber = 1; e**ventID** = LAN1
lanNumber = 2, **eventID** = LAN2
lanNumber = 3; **eventID** = LAN3
lanNumber = 4; **eventID** = LAN4
lanNumber = 5; **eventID** = LAN5
lanNumber = 6; **eventID** = LAN6
lanNumber = 7; **eventID** = LAN7

DESCRIPTION

This function sets the LAN Arm Event ID for a specified LXI event line. This function supports the LxiSync ArmSource interface.

EXAMPLE

vtex10xxA_set_arm_lan_filter

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_arm_lan_filter(ViSession **vi**, ViInt32 **lanNumber**, ViChar **filter[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = a LAN event number corresponding to LXI LAN0 through LAN7 events. Valid input values: 0 through 7.
filter = a LAN Event filter string. See the *LXI Specification* for more detail on filter string syntax.

DATA ITEM RESET VALUE

lanNumber = 0 **filter** = none

DESCRIPTION

This function specifies the LAN arm event filter.

EXAMPLE

vtex10xxA_set_arm_source

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_arm_source (ViSession **vi**, ViInt16 _VI_FAR **vtb_masks[]**,ViInt16 _VI_FAR **dio_masks[]**,ViBoolean **timer_enable**,ViBoolean **immediate**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
timer_enable = a Boolean value indicating whether the timer is enabled as an arm event. If this parameter is set to "1", the timer will act as an arm event. If set to "0", the timer is not a valid arm event.
immediate = a Boolean value indicating whether immediate is enabled as an arm event. If this parameter is set to "1", arm is set to immediate. If a "0" is returned, immediate arm is disabled.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function sets the arm source events. Regardless of this setting, software arms are always enabled. For the **vtb_masks** and **dio_masks** parameters, the data items in these arrays are ordered as follows: positive edge, negative edge, positive level, negative level. Note that , to identify a LAN

EXAMPLE

```
// enable timer arm only
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_source(vi, vtb_masks, dio_masks,1,0);

// enable arm on a positive level on DIO channels 0-3 and a negative edge on VTB channel 6
ViUInt16 vtb_masks[4] = {0,64,0,0};
ViUInt16 dio_masks[4] = {0,0,0x0F,0};
vtex10xxA_set_arm_source(vi, vtb_masks, dio_masks,0,0);

// enable software arm only
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_source(vi, vtb_masks, dio_masks,0, 0);
```

vtex10xxA_set_arm_sourceEx

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_set_arm_sourceEx(ViSession **vi**, ViInt16 **vtb_masks[]**, ViInt16 **dio_masks[]**, ViInt16 **lan_masks[]**, ViBoolean **timer_enable**, ViBoolean **immediate,** ViBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
lan_masks = an array of four 8-bit integer values representing the enabling of LAN events from any of the eight LAN events. Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).
timer = a Boolean value indicating whether the timer is enabled as an arm event. If this parameter is set to "1", the timer will act as an arm event. If set to "0", the timer is not a valid arm event.
immediate = a Boolean value indicating whether immediate is enabled as an arm event. If this parameter is set to "1", arm is set to immediate. If a "0" is returned, immediate arm is disabled.
alarm = a Boolean value indicating whether the LXI alarm is used as an ARM event. If a "1" is set for this parameter, the LXI alarm can act as an arm event. If a "0" is set, the LXI alarm is not a valid arm event.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function sets the arm source events. Regardless of this setting, software arms are always enabled. For the **vtb_masks**, **dio_masks**, and **lan_masks** parameters, the data items in these arrays are ordered as follows: positive edge, negative edge, positive level, negative level. Note that any and all of the parameters above can be enabled simultaneously, with the exception of immediate.

NOTE	It is necessary to explicitly call vtex10xxA_set_alarm_enable to enable alarm events even when the **alarm** parameter is set to TRUE.

EXAMPLE

```
// enable timer arm only
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16 lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 1, 0, 0);

// enable arm on a positive level on DIO channels 0-3 and a negative edge on VTB channel 6
ViUInt16 vtb_masks[4] = {0,64,0,0};
ViUInt16 dio_masks[4] = {0,0,0x0F,0};
ViUInt16 lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks,lan_masks,0, 0, 0);

// enable software arm only
ViUInt16 vtb_masks[4] = {0,0,0,0};
ViUInt16 dio_masks[4] = {0,0,0,0};
ViUInt16 lan_masks[4] = {0,0,0,0};
vtex10xxA_set_arm_sourceEx(vi, vtb_masks, dio_masks, lan_masks, 0, 0, 0);
```

vtex10xxA_set_channel_conversion

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_channel_conversion(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViInt32 **eu_conv**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = the list of channels to which to apply the filter frequency selection. The list of channels can include channels not currently in the scan list and can be a subset of the channels in the scan list.
numChannels = the length of the channels array. Value must be an integer in the range of 1 to 48.
eu_conv = an integer output value representing the EU conversion. Valid return values: 0 through 10. See *Description* for more information.

DATA ITEM RESET VALUE

eu_conv = VTEX10XXA_CONV_MV for all channels

DESCRIPTION

This function sets the EU conversion for the specified channels. For the **eu_conv** parameter, the following values are valid:

Decimal Value	Hex Value	#define	eu_conv Description
0	0x00	VTEX10XXA_CONV_MV	Voltage
1	0x01	VTEX10XXA_CONV_THERMO_TYPE_J	Type J
2	0x02	VTEX10XXA_CONV_THERMO_TYPE_K	Type K
3	0x03	VTEX10XXA_CONV_THERMO_TYPE_T	Type T
4	0x04	VTEX10XXA_CONV_THERMO_TYPE_E	Type E
5	0x05	VTEX10XXA_CONV_THERMO_TYPE_B	Type B
6	0x06	VTEX10XXA_CONV_THERMO_TYPE_S	Type S
7	0x07	VTEX10XXA_CONV_THERMO_TYPE_R	Type R
8	0x08	VTEX10XXA_CONV_THERMO_TYPE_N	Type N
9	0x09	VTEX10XXA_CONV_USER_DEF0	User-defined 0
10	0x0A	VTEX10XXA_CONV_USER_DEF1	User-defined 1

NOTE To meet the accuracy specifications in the *Thermocouple Accuracy* in Table 1-2, the 67 mV range must be used. Using other voltage settings for EU conversion will reduce the accuracy of the measurements. See

EXAMPLE

```
// channels 0-4 are E, channels 5-8 are T
    #define TYPE_E      0x04
    #define TYPE_T      0x03
    ViInt32 e_channels[5] = { 0, 1, 2, 3, 4 };
    vtex10xxA_set_channel_conversion(vi, e_channels, 5, TYPE_E);
    ViInt32 t_channels[4] = { 5, 6, 7, 8 };
    vtex10xxA_set_channel_conversion(vi, t_channels, 4, TYPE_T);
```

vtex10xxA_set_channel_range

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_channel_range(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViReal64 **range**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = the list of channels to which to apply the filter frequency selection. The list of channels can include channels not currently in the scan list and can be a subset of the channels in the scan list.
numChannels = the length of the channels array. Value must be an integer in the range of 1 to 48.
range = sets the input range of the specified channels. Valid input values: 0.01 (10 mV), 0.067 (±67 mV), 0.1 (100 mV), 1 (1 V), and 10 (10 V).

DATA ITEM RESET VALUE

range = 0.067 for thermocouple channels, 10 V for voltage channels.

DESCRIPTION

This function specifies input range of a voltage channel. If the channel specified by the channels parameter is a thermocouple channel, an error will be returned. If an invalid **range** value is used, the value is rounded up to the next highest valid range. If a value is greater than 10.0 V is entered for the **range** parameter, an error will be returned

EXAMPLE

```
//sets an array of the 1032's voltage channels to 1.0 volt range
    ViInt32 ex1032_volt_channels [16] = { 0, 1, 2, 3, 4 ,5 ,6 ,7 ,8 ,9 ,10 ,11 ,12 ,13 ,14 ,15 ,16};
    vtex10xxA_set_channel_ranges(vi, ex1032_volt_channels, 16, 1.0);
```

vtex10xxA_set_communication_timeout

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_communication_timeout (ViSession vi, ViInt32 timeout);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
SETS THE TIMEOUT VALUE (IN MILLISECONDS). INDICATING HOW LONG TO WAIT FOR INSTRUMENT TO RESPOND. IN HIGHER LATENCY NETWORK, IT IS BETTER NOT TO REDUCE DEFAULT TIMEOUT VALUE. VALID RETURN VALUES: 1000 THROUGH (231-1). DEFAULT VALUE: 25000 (25 SECONDS).

DATA ITEM RESET VALUE

Reset will not change value of this parameter.

DESCRIPTION

This function sets the timeout value when a LAN communication error occurs. It is recommended for advanced users only.

EXAMPLE

//sets the LAN communication timeout to 10 seconds, if a LAN communication error occurs
vtex10xxA_set_communication_timeout(vi, 10000);

vtex10xxA_set_dio_limit_event

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_limit_event(ViSession **vi**, ViInt16 **dio_channel**,ViInt16 **limit_masks[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port to be linked to limit evaluations. Value must be an integer in the range of 0 to 7.
limit_masks[] = an array of 48 4-bit integer values representing, on a per input channel basis, the linking of limit evaluations to any of the 4 limit conditions. Within the 4-bit field, the order of the values is: limit set 0 lower, limit set 0 upper, limit set 1 lower, limit set 1 upper. The values must be in the range of 0-15 (decimal), 0x00-0x0F (hex). Channel 0 through channel 47 are represented in array elements [0] through [47], respectively.

DATA ITEM RESET VALUE

limit_masks[] = 0 in all array elements for all DIO channels

DESCRIPTION

This function links limit evaluations to the operation of the digital I/O port. In nominal operation, a DIO channel that is linked to an input channel's limit evaluation will transition from low to high whenever the limit is exceeded. Multiple linkages per DIO channel are allowed and are logically OR'ed together. That is, a DIO channel that is linked to four input channel limit evaluations will transition whenever any of the four limits are exceeded. Multiple linkages can be created on the same input channel and/or spanning multiple input channels.

The nominal transition of a DIO channel is from low to high whenever the linked limit is exceeded. Optionally, the transition can be specified to be from high to low with the vtex10xxA_set_dio_limit_event_invert function.

The nominal operation of a linked DIO channel is to reflect the latest limit evaluation. That is, it will be updated with every scan. Optionally, the behavior can be set to latch mode with the vtex10xxA_set_dio_limit_event_latch function . In latch mode, a transition out of the cleared state would remain, regardless of future limit evaluations, until it is cleared at the beginning of a new acquisition.

EXAMPLE

```
// link DIO channel 7 to set 1 lower limits on ch 12 and 14, set 0 upper limit on ch 1
ViUInt16 limit_masks[48] = {0,4,0,0,0,0,0,0,0,0,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

vtex10xxA_set_dio_limit_event(vi, 7, limit_masks);
```

vtex10xxA_set_dio_limit_event_invert

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_limit_event_invert(ViSession **vi**, ViInt16 **dio_channel**, ViBoolean **invert**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port to be affected. Valid input values: 0 through 7.
invert = a Boolean value indicating whether to operate in invert mode. If set to “1”, the specified DIO channel will function in invert mode. If set to “0”, DIO operation is not inverted.

DATA ITEM RESET VALUE

invert = 0 (for all DIO channels)

DESCRIPTION

This function enables or disables inverted operation of a DIO channel linked as a limit event. The nominal transition of a DIO channel is from low to high whenever the linked limit is exceeded. In invert mode, it is from high to low.

EXAMPLE

vtex10xxA_set_dio_limit_event_latch

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_limit_event_latch(ViSession **vi, ViInt16 dio_channel, ViBoolean latch**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_channel = the channel of the digital I/O port to be affected. Valid input values: 0 through 7.
latch = a Boolean value indicating whether to operate in latch mode. If this parameter is set to “1”, the specified DIO channel will operate in latch mode. If set to “0”, the specified DIO latch mode is not enabled.

DATA ITEM RESET VALUE

latch = 0 (for all DIO channels)

DESCRIPTION

This function enables or disables latch operation of a DIO channel linked as a limit event. The nominal operation of a linked DIO channel is to reflect the latest limit evaluation. That is, it will be updated with every scan. In latch mode, a transition out of the cleared state would remain, regardless of future limit evaluations, until it is cleared at the beginning of a new acquisition.

EXAMPLE

vtex10xxA_set_dio_output

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_output(ViSession **vi**, ViInt32 **dio_out**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_out = the value that represents the desired state of the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

dio_out = 0

DESCRIPTION

This function sets the static level that each channel of the digital I/O port will assume if enabled. Enabling is done with the vtex10xxA_set_dio_output_enable function.

EXAMPLE

```
// set DIO bit 7 (high) and DIO bit 6 (low)
        vtex10xxA_set_dio_output(vi, 0x80);
// enable them as outputs
        vtex10xxA_set_dio_output_enable(vi, 0xC0);
```

vtex10xxA_set_dio_output_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_output_enable(ViSession **vi**, ViInt32 **out_enable**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
out_enable = the value that represents the desired output enable state of the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

dio_enable = 0

DESCRIPTION

This function enables or disables the output functionality of each channel of the digital I/O port. Input functionality on each channel is constant regardless of its output functionality.

EXAMPLE

```
// set DIO bit 7 (high) and DIO bit 6 (low)
        vtex10xxA_set_dio_output(vi, 0x80);
// enable them as outputs
        vtex10xxA_set_dio_output_enable(vi, 0xC0);
```

vtex10xxA_set_dio_pulse

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_dio_pulse(ViSession **vi**, ViInt32 **dio_pulse**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
dio_pulse = the value that represents the channels to be pulsed within the 8-bit port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function generates a 1 µs pulse on selected channels of the digital I/O port. The pulse will occur only if the selected channels are enabled as outputs. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going.

EXAMPLE

```
// set DIO bit 7 low and then pulse high
    vtex10xxA_set_dio_output(vi, 0x00);
    vtex10xxA_set_dio_output_enable(vi, 0x80);
    vtex10xxA_set_dio_pulse(vi, 0x80);
```

vtex10xxA_set_fifo_config

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_fifo_config(ViSession **vi**, ViBoolean **report_cjc**, ViBoolean **report_timestamp**, ViBoolean **report_celsius**, ViBoolean **blocking_mode**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
report_cjc = a Boolean value indicating whether to report the CJC temperatures in the acquisition data. If set to “0”, CJC temperatures will not be reported, while setting this parameter to “1” enables reporting.
report_timestamp = a Boolean value indicating whether to report delta timestamps per channel in the acquisition data. When this parameter is set to “1”, timestamp reporting is enabled. When set to “0”, it is disabled.
report_celsius = a Boolean value indicating whether to report the input channel data in units of Celsius (ºC) or Fahrenheit (ºF). If this parameter is set to “1”, input channel data is reported in °C. If set to “0”, data is reported in °F
blocking_mode = a Boolean value that indicates whether blocking mode is enabled. When this parameter is set to “1”, blocking mode is enabled. When set to “0”, blocking mode is disabled.

DATA ITEM RESET VALUE

report_cjc = 0
report_timestamp = 0
report_celsius = 1
blocking_mode = 0

DESCRIPTION

This function sets the data format and overflow behavior of the FIFO memory.

The **report_cjc** parameter does not affect the measurement of CJC temperature, but determines if the data will be displayed. CJC temperatures are measured with every scan, regardless of this setting. Reported CJC temperatures are always in units of ºC, regardless of the value of the **report_celsius** parameter.

The **report_timestamp** parameter represents the time, in increments of 100 ns, between the specific channel measurement and the scan initiation time.

The **blocking_mode** parameter sets defines the action taken if the reading buffer fills to its maximum capacity during scanning. When this mode is enabled, additional readings are discarded leaving the contents of the buffer intact. With blocking mode disabled, the buffer becomes circular, in that additional readings overwrite the oldest readings.

NOTE	Regardless of the settings of the **report_cjc** and **report_timestamp** parameters, the CJC temperature and delta timestamp information is not accessible through the vtex10xxA_read_fifo function. They are available through the streaming interface and the parameter settings apply for it.

EXAMPLE

vtex10xxA_set_filt_freq

FUNCTION PROTOTYPE

ViStatus_VI_FUNC vtex10xxA_set_filt_freq(ViSession **vi**, ViInt32 **channels[]**,ViInt32 **numChannels**, ViReal64 **filt_freq**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels[] = the list of channels to which to apply the filter frequency selection. The list of channels can include channels not currently in the scan list and can be a subset of the channels in the scan list. Valid input values: 0 through 47.
numChannels = the length of the channels array. Valid input values: 1 through 48.
filt_freq = a real output value representing the frequency setting in hertz (Hz). Valid input values: 4, 15, 40, 100, 500, and 1000.

DATA ITEM RESET VALUE

filt_freq = 4.0 (for all channels)

DESCRIPTION

This function sets low-pass filter cutoff frequency of specified channels. Supported frequencies are 4 Hz, 15 Hz, 40 Hz, 100 Hz, 500 Hz, and 1 kHz. If a value other than the valid frequencies are used, this function round up to the next highest valid frequency. If a number greater than 1000 is used, the filter is set to 1 kHz.

NOTE	For the EX1048, the only available frequencies are 4 Hz and 1 kHz.

EXAMPLE

```
// set channels 0-4 for 4 Hz
        ViInt32 low_channels[5] = { 0, 1, 2, 3, 4 };
        vtex10xxA_set_filt_freq(vi, low_channels, 5, 4.0);

// set channels 5-8 for 1 kHz
        ViInt32 high_channels[4] = { 5, 6, 7, 8 };
        vtex10xxA_set_filt_freq(vi, high_channels, 4, 1000.0);
```

vtex10xxA_set_init_cont

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_init_cont(ViSession **vi**, ViBoolean **init_cont_mode**):

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
init_cont_mode = a Boolean value indicating whether to enable init continuous mode. If this parameter is set to “1”, init continuous mode is enabled. If set to “0”, init continuous mode is disabled.

DATA ITEM RESET VALUE

init_cont_mode = 0

DESCRIPTION

This function enables or disables the use of init continuous mode. Init continuous returns the trigger model to the entrance of the ARM layer without the requirement of a new trigger initialize command.

EXAMPLE

vtex10xxA_set_lan_event_domain

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_lan_event_domain (ViSession **vi**, ViInt32 **domain**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
domain = a integer value indicating the instrument's LAN Event domain. Valid input values: 0 through 255.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function sets the instrument's LAN Event domain. The domain property is persistent across reboots and resets.

EXAMPLE

vtex10xxA_set_lan_eventlog_enabled

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_set_lan_eventlog_enabled(ViSession **vi**, ViBoolean **enable**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
enable = a Boolean value which sets the status of event logging. If this parameter is set to “1”, event logging is enabled. If set to “0”, event logging is disabled.

DATA ITEM RESET VALUE

enable = 0

DESCRIPTION

This function enables/disables event log. This function supports the LxiSync EventLog interface.

EXAMPLE

vtex10xxA_set_lan_eventlog_overflowmode

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_lan_eventlog_overflowmode (ViSession **vi**,ViInt32 **mode**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
mode = integer value that indicates whether the oldest event log shall be overwritten (VTEX10XXA_EVENTLOG_OVERFLOWMODE_OVERWRITE) or the new event log shall be ignored (VTEX10XXA_EVENTLOG_OVERFLOWMODE_IGNORE) when the event log buffer is full.

DATA ITEM RESET VALUE

mode = 0

DESCRIPTION

This function sets the LAN event log overflow mode. In the event that the LAN event log buffer becomes full, the EX10xxA will either overwrite the existing log or will not write the new log information.

The **mode** parameter has the following valid input values:

Decimal Value	#define Symbol	mode Description
0	VTEX10XXA_EVENTLOG_OVERFLOWMODE_OVERWRITE	Overwrites oldest event log buffer.
1	VTEX10XXA_EVENTLOG_OVERFLOWMODE_IGNORE	Ignores new event log data.

EXAMPLE

vtex10xxA_set_limit_set0

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_limit_set0(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViReal64 **lower_limit**, ViReal64 **upper_limit**)

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
lower_limit = sets the value for the lower reading limit. The limit will be tripped if the applicable channel reading is less than the lower reading limit. For proper operation, the limit value must be entered in the same units as the channels with which it is associated. Valid input values: -6.6e-2 through 6.6e-2.
upper_limit = the value for the upper reading limit. The limit will be tripped if the applicable channel reading is greater than the upper reading limit. For proper operation, the limit value must be entered in the same units as the channels with which it is associated. Valid input values: -6.6e-2 through 6.6e-2.

DATA ITEM RESET VALUE

lower_limit = -6.6e-2 (for all channels) **upper_limit** = 6.6e-2 (for all channels)

DESCRIPTION

This function sets the limit set 0 values manually for the specified channels. A channel's limit values can be set regardless of its inclusion in the scan list, and multiple channels can be assigned to the same limit values with a single function call. However, each unique combination of limit values must be set with a separate function call.

By default, the limit values for limit set 0 are set automatically, based on the EU conversion and units selection for each channel. If manual limit control has been enabled with the vtex10xxA_set_limit_set0_manual function, user defined limit values can be entered. If manual limit control is not enabled, execution of this function will not generate an error, but the specified limit values will be ignored.

For proper operation, limit values must be entered and maintained in the same units as the channels with which they are associated. Once entered, manual limit values are not automatically converted by subsequent changes in EU conversion or units designations.

EXAMPLE

```
// set channels 0-4 to manual limits of 0 and 100
ViInt32 e_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_limit_set0_manual(vi, e_channels, 5, 1);
vtex10xxA_set_limit_set0(vi, e_channels, 5, 0, 100);
```

vtex10xxA_set_limit_set0_manual

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_limit_set0_manual(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels, ViBoolean manual**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
manual = a Boolean value setting the status of manual limit value entry. If this parameter is set to “1”, manual limit value entry is enabled. If set to “0”, manual limit value entry is disabled.

DATA ITEM RESET VALUE

manual = 0 (for all channels)

DESCRIPTION

This function enables or disables manual entry of limit set 0 values for the specified channels. A channel’s manual entry control can be set regardless of its inclusion in the scan list, and multiple channels can be configured with a single function call. However, each unique control value must be set with a separate function.

By default, the limit values for limit set 0 are set automatically, based on the EU conversion and units selection for each channel. If manual limit control is desired, it must first be enabled with this function.

Disabling manual limit control on a channel that had been enabled will automatically set the limit set 0 values for that channel, based on its current EU conversion and units selection.

EXAMPLE

vtex10xxA_set_limit_set1

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_limit_set1(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViReal64 **lower_limit**, ViReal64 **upper_limit**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
lower_limit = sets the value for the lower reading limit. The limit is tripped if the applicable channel reading is less than the lower reading limit. For proper operation, the limit value must be entered in the same units as the channels with which it is associated. Valid input values: -3e+38 through 3e+38.
upper_limit = the value for the upper reading limit. The limit is tripped if the applicable channel reading is greater than the upper reading limit. For proper operation, the limit value must be entered in the same units as the channels with which it is associated. Valid input values: -3e+38 through 3e+38.

DATA ITEM RESET VALUE

lower_limit = -3e+38 (for all channels)

upper_limit = -3e+38 (for all channels)

DESCRIPTION

This function sets the limit set 1 values for the specified channels. A channel's limit values can be set regardless of its inclusion in the scan list, and multiple channels can be assigned to the same limit values with a single function call. However, each unique combination of limit values must be set with a separate function call.

Unlike limit set 0, the limit values for limit set 1 are always set manually.

For proper operation, limit values must be entered and maintained in the same units as the channels with which they are associated. Once entered, limit values are not automatically converted by subsequent changes in EU conversion or units designations.

EXAMPLE

vtex10xxA_set_linear_correction

FUNCTION PROTOTYPE

ViStatus VI_FUNC vtex10xxA_set_linear_correction(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViReal64 **gain**, ViReal64 **offset**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
gain = a real input value representing the gain component, **m**, of linear EU correction, m(x-b). Valid input values: all real numbers.
offset = a real input value representing the offset component, **b**, of linear EU correction m(x-b). Valid input values: all real numbers.

DATA ITEM RESET VALUE

For all channels:
gain = 1.0 **offset** = 0.0

DESCRIPTION

This function specifies m(x-b) linear correction of a voltage channel.

EXAMPLE

vtex10xxA_set_OTD_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_OTD_enable(ViSession **vi**, ViInt32 _VI_FAR **channels[]**,ViInt32 **numChannels**, ViBoolean **otd**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels[]= an array of channels where OTD will be enabled. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the **channels** array. Valid input values: 1 through 48.
otd = a Boolean value indicating whether to enable the open transducer detection.

DATA ITEM RESET VALUE

numChannels = 0 **otd** = 0

DESCRIPTION

This function enables or disables the use of OTD (Open Transducer Detection) for the specified channels.

NOTE	OTD does not work on the 10 V range.

EXAMPLE

vtex10xxA_set_scanlist

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_scanlist(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.

DATA ITEM RESET VALUE

channels = 0 **numChannels** = 1

DESCRIPTION

This function sets the scan list to be acquired. For the channels parameter, the elements of the array must be unique (i.e. no repeated channels). The channels parameter should be set to an integer between 1 and 48.

vtex10xxA_get_scanlist is called, data is returned in ascending numerical order by channel number.

EXAMPLE

```
// sequential order five-channel scan
        ViInt32 channels[5] = { 0, 1, 2, 3, 4 };
        vtex10xxA_set_scanlist(vi, channels, 5);

// reverse order five-channel scan
        ViInt32 channels[5] = { 4, 3, 2, 1, 0 };
        vtex10xxA_set_scanlist(vi, channels, 5);
```

vtex10xxA_set_trig_lan_eventID

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_trig_lan_eventID(ViSession **vi**, ViInt32 **lanNumber**, ViChar **eventID[])**;

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = an integer input value indicating the LAN event number corresponding to LAN0 through LAN7. Valid input values: 0 through 7.
eventID = the Trigger LAN event ID string for the specified LAN Event. Valid input values: ASCII strings of length 1 through 16.

DATA ITEM RESET VALUE

lanNumber = 0; **eventID** = LAN0
lanNumber = 1; **eventID** = LAN1
lanNumber = 2; **eventID** = LAN2
lanNumber = 3; **eventID** = LAN3
lanNumber = 4; **eventID** = LAN4
lanNumber = 5; **eventID** = LAN5
lanNumber = 6; **eventID** = LAN6
lanNumber = 7; **eventID** = LAN7

DESCRIPTION

This function specifies event ID of LAN trigger event. This function supports the LxiSync TriggerSource interface. The valid **lanNumber**s, 0 through 7, correspond to LXI LAN0 through LAN7 events.

EXAMPLE

vtex10xxA_set_trig_lan_filter

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_trig_lan_filter(ViSession **vi**, ViInt32 **lanNumber**, ViChar **filter[]**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
lanNumber = an integer input value indicating the LAN event number corresponding to LAN0 through LAN7. Valid input values: 0 through 7.
filter = a LAN Event filter string.. Valid input values: ASCII strings of length 0 through 512. See the *IVI-3.15 IVILXISync* specification for full filter string syntax.

DATA ITEM RESET VALUE

filter = “” (for all LAN events)

DESCRIPTION

This function specifies event ID of LAN trigger event. This function supports the LxiSync TriggerSource interface. Valid **lanNumber** values are 0 through 7 and correspond to LXI LAN0 through LAN7 events.

EXAMPLE

vtex10xxA_set_trig_source_timer

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trig_source_timer(ViSession **vi**, ViReal64 **trig_timer**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
trig_timer = the time value (in seconds) between timer ticks. Valid input values: 0.001 (1 ms) through 4294 (71.5 minutes) with a resolution of 0.000001 (1 µs).

DATA ITEM RESET VALUE

trig_timer = 0.1

DESCRIPTION

This function sets a trigger source of timer only and sets the timer interval. This is a one-function combination of the vtex10xxA_set_trigger_sourceEx and vtex10xxA_set_trigger_timer functions.

EXAMPLE

vtex10xxA_set_trigger_count

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_count(ViSession **vi**, ViInt32 **count**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
count = sets the trigger count value. Valid input values: 1 through (2^{31}-1).

DATA ITEM RESET VALUE

count = 1

DESCRIPTION

This value sets the trigger count value. This value is reset with each arm event.

EXAMPLE

```
// set a trigger count of 10
vtex10xxA_set_trigger_count(vi, 10);
```

vtex10xxA_set_trigger_delay

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_delay(ViSession **vi**, ViReal64 **trig_delay**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
trig_delay = sets the delay value (in seconds). Valid input values: 0 through 4294 (71.5 minutes) with a resolution of 0.000001 (1 µs).

DATA ITEM RESET VALUE

trig_delay = 0

DESCRIPTION

This function sets the trigger delay, the time between the recognition of the trigger event and the execution of the scan list.

EXAMPLE

```
// set a trigger delay of 5 ms
vtex10xxA_set_trigger_delay(vi, 0.005);
```

vtex10xxA_set_trigger_infinite

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_infinite(ViSession **vi**, ViBoolean **triginf**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
triginf = a Boolean value which determines whether trigger count is set to infinite. If this parameter is set to “1”, the trigger count is set to infinite. If set to “0”, the trigger count is not infinite.

DATA ITEM RESET VALUE

triginf = 0

DESCRIPTION

This function enables or disables the use of an infinite trigger count. Enabling overrides any manual setting of trigger count.

EXAMPLE

vtex10xxA_set_trigger_source

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_source(ViSession **vi**,ViInt16 **vtb_masks[]**,ViInt16 **dio_masks[]**,ViBoolean **timer_enable**,ViBoolean **immediate**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
timer_enable = a Boolean value indicating whether the timer is enabled as a trigger event. If this parameter is set to "1", the timer is enabled as a trigger. If set to "0", timer triggering is disabled.
immediate = a Boolean value indicating whether immediate is enabled as a trigger event. If this parameter is set to "1", immediate triggering is enabled. If set to "0", immediate triggering is disabled.

DATA ITEM RESET VALUE

The trigger source is set to timer.

DESCRIPTION

This function sets the trigger source events. Regardless of this setting, software triggers are always enabled.

EXAMPLE

vtex10xxA_set_trigger_sourceEx

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_sourceEx(ViSession **vi**, ViInt16 **vtb_masks[]**, ViInt16 **dio_masks[]**, ViInt16 lan_masks[], ViBoolean **timer_enable**, ViBoolean **immediate,** ViBoolean **alarm**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the trigger bus. Within the 8-bit field, the MSB corresponds to VTB channel 7 and the LSB corresponds to VTB channel 0. Valid input values: 0 through 255.
dio_masks = a return array of four 8-bit integer values representing the enabled state of trigger events from the eight channels of the digital I/O port. Within the 8-bit field, the MSB corresponds to DIO channel 7 and the LSB corresponds to DIO channel 0. Valid input values: 0 through 255.
lan_masks = an array of four 8-bit integer values representing the enabling of LAN events from any of the eight LAN events. Within the 8-bit field, the MSB corresponds to LAN 7 and the LSB corresponds to LAN 0. Valid input values: 0 (0x00h) through 255 (0xFFh).
timer = a Boolean value indicating whether the timer is enabled as a trigger event. If this parameter is set to “1”, the timer is enabled as a trigger. If set to “0”, timer triggering is disabled.
immediate = a Boolean value indicating whether immediate is enabled as a trigger event. If this parameter is set to “1”, immediate triggering is enabled. If set to “0”, immediate triggering is disabled.
alarm = a Boolean value indicating whether the LXI alarm is used as an ARM event. If a “1” is set for this parameter, the LXI alarm can act as an arm event. If a “0” is set, the LXI alarm is not a valid arm event.

DATA ITEM RESET VALUE

The trigger source is set to timer.

DESCRIPTION

This function sets the trigger source events. Regardless of this setting, software triggers are always enabled.

NOTE	It is necessary to explicitly call vtex10xxA_set_alarm_enable to enable timer events even when the **alarm** parameter is set to TRUE.

EXAMPLE

vtex10xxA_set_trigger_timer

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_trigger_timer (ViSession **vi**, ViReal64 **trig_timer**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
trig_timer = the amount of time, in seconds, between timer ticks. Valid input values: 0.001 (1 ms) to 4294 (71.5 minutes) with a resolution of 1 µs.

DATA ITEM RESET VALUE

trig_timer = 0.1

DESCRIPTION

This function sets the timer interval for a timer source event. The same value is used for both arm and trigger events.

EXAMPLE

vtex10xxA_set_user_cjc_enable

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_user_cjc_enable(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels, ViBoolean usercjc**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
usercjc = a Boolean value indicating whether to enable the user-defined CJC temperature. If this parameter is set to "1", this indicates that the user-defined CJC temperature will be used in thermocouple calculations. If set to "0", this functionality is disabled.

DATA ITEM RESET VALUE

usercjc = 0 (for all channels)

DESCRIPTION

This function enables or disables the use of a user-defined CJC temperature for the specified channels. If enabled, the user-defined CJC temperature will be used in the thermocouple calculations instead of the internally measured one. Each channel can be associated with a unique value and be independently enabled with regards to its use. The entry of external CJC values and their enabling are disjoint functions. That is, the entry of a value does not automatically enable its use, and the disabling of a previously enabled channel does not clear the value. Entry of the CJC temperature is done through the vtex10xxA_set_user_cjc_temp function.

NOTE	This function should only be used when the thermocouple cold junction is made external to the EX10xxA. This feature must be used with care, as the input channel data contains no indication that it was calculated with an external CJC value instead of an internal one.

EXAMPLE

```
// CJC for channels 0-4 are held externally at 0.2 C
ViInt32 ext_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_user_cjc_temp(vi, ext_channels, 5, 0.2);
vtex10xxA_set_user_cjc_enable(vi, ext_channels, 5, 1);
```

vtex10xxA_set_user_cjc_temp

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_user_cjc_temp(ViSession **vi**, ViInt32 **channels[]**, ViInt32 **numChannels**, ViReal64 **cjc_temp**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
channels = an array to receive the requested scan list. Valid input values: 0 through 47.
numChannels = an integer output value indicating how many channels are in the scan list. Valid input values: 1 through 48.
cjc_temp = sets the value of the user-defined CJC temperature. The value is entered with units of ºC, regardless of the units selection of the input channels. To ensure accuracy, J, K, T, E, and N types should be set to values between -40 and 75 (inclusive), while B, R, and S types should be set to values between 0 and 75 (inclusive). Any floating-point number, however, will be accepted by this parameter.

DATA ITEM RESET VALUE

cjc_temp = 0 (for all channels)

DESCRIPTION

This function sets the user-defined CJC temperature for the specified channels. If enabled, the temperature entered will be used in thermocouple calculations instead of the internally measured one. Each channel can be associated with a unique value and be independently enabled with regards to its use. The entry of external CJC values and their enabling are disjoint functions. That is, the entry of a value does not automatically enable its use, and the disabling of a previously enabled channel does not clear the value. Enabling is done through the vtex10xxA_set_user_cjc_enable function.

NOTE	This function should only be used when the thermocouple cold junction is made external to the EX10xxA. This feature must be used with care, as the input channel data contains no indication that it was calculated with an external CJC value instead of an internal one.

EXAMPLE

```
// CJC for channels 0-4 are held externally at 0.2 C
ViInt32 ext_channels[5] = { 0, 1, 2, 3, 4 };
vtex10xxA_set_user_cjc_temp(vi, ext_channels, 5, 0.2);
vtex10xxA_set_user_cjc_enable(vi, ext_channels, 5, 1);
```

vtex10xxA_set_user_conversion

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_user_conversion(ViSession **vi**, ViInt32 **eu_conv**, ViReal64 **fwdcoeff[]**, ViInt32 **numFwd**, ViReal64 **invcoeff[]**, ViInt32 **numInv**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
eu_conv = the polynomial set to be defined. Value must be an integer equal to 9 (User0) or 10 (User1).
fwdcoeff[] = an array of forward conversion polynomial coefficients. Coefficients c_0 through c_{12} are represented in array elements [0] through [12], respectively.
numFwd = the length of the forward coefficients array. Valid input values: 1 through 13.
invcoeff[] = an array of inverse conversion polynomial coefficients. Coefficients d_0 through d_{12} are represented in array elements [0] through [12], respectively.
numInv = the length of the inverse coefficients array. Valid input values: 1 through 13.

DATA ITEM RESET VALUE

fwdcoeff[] = 0 in all array elements for both polynomial sets.
invcoeff[] = 0 in all array elements for both polynomial sets.

DESCRIPTION

This function sets the user-defined conversion polynomials.

The forward conversion polynomial is used to convert a CJC temperature into a compensating cold junction voltage and has the form of:

$$E = c_0 + c_1 * t^1 + c_2 * t^2 + \ldots + c_{12} * t^{12}$$

where E is in volts, t is in ºC, and $c_o - c_{12}$ are the coefficients.

The inverse conversion polynomial is used to convert a compensated input voltage into temperature and has the form of:

$$t = d_0 + d_1 * E^1 + d_2 * E^2 + \ldots + d_{12} * E^{12}$$

where E is in volts, t is in ºC, and $d_o - d_{12}$ are the coefficients.

Undefined coefficients are automatically set to 0.

NOTE	The entry of user-defined coefficients does not automatically enable their use. The enabling is done by setting the EU conversion to User0 or User1 through the vtex10xxA_set_channel_conversion function.

EXAMPLE

vtex10xxA_set_vtb_output

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_vtb_output(ViSession **vi**, ViInt32 **vtb_out**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_out = the value that represents the desired state of the 8-bit port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

vtb_out = 0

DESCRIPTION

This function sets the static level that each channel of the trigger bus will assume if enabled. Enabling is done with the vtex10xxA_set_vtb_output_enable function.

EXAMPLE

```
// set VTB bit 7 (high) and VTB bit 6 (low)
        vtex10xxA_set_vtb_output(vi, 0x80);
// enable them as outputs
        vtex10xxA_set_vtb_output_enable(vi, 0xC0);
```

vtex10xxA_set_vtb_output_enable

FUNCTION PROTOTYPE

vtex1048_set_vtb_output_enable(ViSession **vi**, ViInt32 **out_enable**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
out_enable = the value that represents the desired output enable state of the 8-bit port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

out_enable = 0

DESCRIPTION

This function enables or disables the output functionality of each channel of the trigger bus. Input functionality on each channel is constant regardless of its output functionality.

EXAMPLE

```
// set VTB bit 7 (high) and VTB bit 6 (low)
        vtex10xxA_set_vtb_output(vi, 0x80);
// enable them as outputs
        vtex10xxA_set_vtb_output_enable(vi, 0xC0);
```

vtex10xxA_set_vtb_pulse

FUNCTION PROTOTYPE

vtex1048_set_vtb_pulse(ViSession **vi**, ViInt32 **vtb_pulse**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
vtb_pulse = the value that represents the channels to be pulsed within the 8-bit port. Within the 8-bit field, the MSB corresponds to LXI Trigger Bus (VTB) channel 7 and the LSB corresponds to LXI Trigger Bus (VTB) channel 0. Valid input values: 0 through 255 (decimal), 0x00 through 0xFF (hexadecimal).

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function generates a 1 µs pulse on selected channels of the trigger bus. The pulse will occur only if the selected channels are enabled as outputs. When a channel is programmed with a static level of high, the pulse will be low-going. When a channel is programmed with a static level of low, the pulse will be high-going.

EXAMPLE

vtex10xxA_set_vtb_wiredor

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_vtb_wiredor (ViSession **vi**,ViInt32 **wiredor**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
wiredor = an integer input value that indicates the Wired-OR state for all eight LXI Trigger Bus lines. Only the least significant bits are used. The least significant bit of the integer represents LXI Trigger Bus line 0, where the 8th least significant bit corresponds to LXI Trigger Bus line 7. A return value of "1" for any of the eight bits indicates that the wired or mode is enabled for that particular LXI Trigger Bus line, while a value of "0" indicates that Wired-OR is disabled.

DATA ITEM RESET VALUE

wiredor = 0

DESCRIPTION

This function sets the Wired-OR state for each of the individual LXI Trigger Bus lines.

EXAMPLE

vtex10xxA_set_vtb_wiredor_bias

FUNCTION PROTOTYPE

ViStatus vtex10xxA_set_vtb_wiredor_bias (ViSession **vi**, ViInt32 **wiredor_bias**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
wiredor_bias = an integer input value that indicates the Wired-OR bias enabled state for all eight LXI Trigger Bus lines. Only the least significant bits are used. The least significant bit of the integer represents LXI Trigger Bus line 0, where the 8th least significant bit corresponds to LXI Trigger Bus line 7. A value of "1" value for any of the eight bits indicates that the wired or bias mode is enabled for the particular LXI trigger bus line, while a "0" indicates that it is disabled.

DATA ITEM RESET VALUE

wiredor_bias = 0

DESCRIPTION

This function sets the Wired-OR bias enable state for each LXI Trigger Bus lines.

EXAMPLE

vtex10xxA_soft_arm

FUNCTION PROTOTYPE

ViStatus vtex10xxA_soft_arm(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs a software arm. Software arms are always enabled, regardless of the state of the arm source.

EXAMPLE

vtex10xxA_soft_trigger

FUNCTION PROTOTYPE

ViStatus vtex10xxA_soft_trigger(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs a software trigger. Software triggers are always enabled, regardless of the state of the trigger source.

EXAMPLE

vtex10xxA_unlock

FUNCTION PROTOTYPE

ViStatus vtex10xxA_unlock(ViSession **vi**);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function releases a lock on the instrument. It can be successfully executed only by the host IP address that originally acquired the lock. In order to break a lock owned by another user, the vtex10xxA_break_lock function must be used.

As the lock status of the instrument is unaffected by the vtex10xxA_reset function, it cannot be used to release a lock.

EXAMPLE

vtex10xxA_get_shield_gnd_config

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_shield_gnd_config (ViSession vi, ViPInt32 shield_gnd);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
shield_gnd = an integer output indicating to receive shield ground information from the instrument. Valid return values: 0 through 2

VTEX10XXA_SHLD_GND_NONE = 0
VTEX10XXA_SHLD_GND_SIGNAL = 1
VTEX10XXA_SHLD_GND_CHASSIS = 2

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function is to get whether the shield connected to Signal Ground, Chassis or left disconnected. This is only supported for RX10XX series instruments.

EXAMPLE

vtex10xxA_set_communication_timeout

FUNCTION PROTOTYPE

ViStatus _VI_FUNC vtex10xxA_set_communication_timeout (ViSession vi, ViInt32 timeout);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
SETS THE TIMEOUT VALUE (IN MILLISECONDS). INDICATING HOW LONG TO WAIT FOR INSTRUMENT TO RESPOND. IN HIGHER LATENCY NETWORK, IT IS BETTER NOT TO REDUCE DEFAULT TIMEOUT VALUE. VALID RETURN VALUES: 1000 THROUGH (231-1). DEFAULT VALUE: 25000 (25 SECONDS).

DATA ITEM RESET VALUE

Reset will not change value of this parameter.

DESCRIPTION

This function sets the timeout value when a LAN communication error occurs. It is recommended for advanced users only.

EXAMPLE

```
//sets the LAN communication timeout to 10 seconds, if a LAN communication error occurs
vtex10xxA_set_communication_timeout(vi, 10000);
```

vtex10xxA _get_utc_offset

FUNCTION PROTOTYPE

ViStatus vtex10xxA_get_utc_offset (ViSession vi, ViPInt32 UTCOffset);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
timeout = an integer value that indicates current UTC offset from master clock. Zero if there is no PTP master clock source
VALID RETURN VALUES: Between 0 and 35 and change as per UTC offset value which will be set based on a calendar year.

DATA ITEM RESET VALUE

Reset will not change value of this parameter.

DESCRIPTION

This function gets the UTC offset from master PTP clock source.

EXAMPLE

vtex10xxA_self_cal_init_ex

FUNCTION PROTOTYPE

ViStatus vtex10xxA_self_cal_init_ex (ViSession vi, ViPInt32 override, ViPInt32 current_uptime);

FUNCTION PARAMETERS

vi = contains a session handle to the instrument. This handle is obtained by the function and remains valid until the session is closed.
override = an integer output value in decimal that represents an override code. If self-calibration is attempted before the EX10xxA has been powered on continuously for 60 minutes, an error will be generated, and an integer value will be placed in the **override** variable. If appropriate, resending the function will override the error and initiate self-calibration.
current_uptime = an integer output value in decimal that represents the time elapsed in seconds since the instrument is turned on.

DATA ITEM RESET VALUE

Not applicable to this function.

DESCRIPTION

This function performs an instrument self-calibration. In general, self-calibration should not be performed until the instrument has been powered on continuously for 60 minutes. In fact, if self-calibration is attempted prior to the required uptime, an error will be generated. When appropriate, as described below, this error can be overridden by resending the function.

NOTES

1) In order to perform a self-calibration, a lock on the instrument must first be acquired. Attempting to self-calibrate without the acquisition of a lock will generate an error that is not able to be overridden. See the vtex10xxA_lock function.

2) The self-calibration uptime requirement is in place to protect the measurement integrity of the instrument. Overriding the requirement must only be done when the operating conditions allow it. An example of this is where the unit has actually been warmed up, but has simply been subjected to a quick power cycle or reboot. **In order to ensure that the override is intentional, it is strongly recommended that user intervention be required in the software application to employ it.**

3) Once self-calibration has been successfully initiated, its percentage completion status is accessible through the vtex10xxA_self_cal_get_status function. **Additional instrument driver calls should not be performed until the completion status reaches 100 percent.** To determine if calibration is still running, use the **vtex10xxA_get_calibration_running** function. If calibration fails, it status will not reach 100%.

EXAMPLE

ERROR MESSAGES

Each function in this instrument driver returns a status code that either indicates success or describes an error or warning condition. Programs should examine the status code from each call to an instrument driver function to determine if an error occurred. The general meaning of the status code is as follows:

Value	Meaning
VI_SUCCESS (0)	Success
Positive Values	Warnings
Negative Values	Errors

The following table lists possible error codes and their meanings which may be returned by driver functions. The following error codes, in hexadecimal, may be encountered while operating the EX10xxA. The corresponding error messages' text can be obtained by using the vtex10xxA_error_message function.

Error Code	Error Message	Meaning
0xBFFC0800L	#define VTEX10XXA_ERROR_INSTR_FILE_OPEN	The driver failed to read a file.
0xBFFC0801L	#define VTEX10XXA_ERROR_INSTR_FILE_WRITE	The driver failed to write a file.
0xBFFC0803L	#define VTEX10XXA_ERROR_INSTR_INTERPRETING_RESPONSE	The driver did not understand the instrument's response.
0xBFFC08F0L	#define VTEX10XXA_ERROR_RUNTIME	Runtime error.
0xBFFC08F1L	#define VTEX10XXA_ERROR_INVALID_CONFIGURATION	Invalid configuration.
0xBFFC08F2L	#define VTEX10XXA_ERROR_ID_MISMATCH	ID mismatch.
0xBFFC08F3L	#define VTEX10XXA_ERROR_ID_QUERY	ID query.
0xBFFC08F4L	#define VTEX10XXA_ERROR_MAX_INSTRUMENT	Max instrument exceeded.
0xBFFC08F5L	#define VTEX10XXA_ERROR_CLNT_CREATE	This driver does not support the device indicated by the provided resource string.
0xBFFC08F6L	#define VTEX10XXA_ERROR_SYNCHRONIZATION	Synchronization error.
0xBFFC08F7L	#define VTEX10XXA_ERROR_MEM_ALLOC	Memory allocation.
0xBFFC08F8L	#define VTEX10XXA_ERROR_RPC	RPC error.
0xBFFC08F9L	#define VTEX10XXA_ERROR_INVALID_LIST	Invalid scanlist number.
0xBFFC08FAL	#define VTEX10XXA_ERROR_INVALID_CHANNEL	Invalid channel number.
0xBFFC08FBL	#define VTEX10XXA_ERROR_INVALID_NUM_CHANNELS	Invalid number of channels.
0xBFFC08FCL	#define VTEX10XXA_ERROR_INVALID_SAMPLE_PERIOD	Invalid sample period.
0xBFFC08FDL	#define VTEX10XXA_ERROR_INVALID_EU_CONV	Invalid engineering unit (EU) conversion
0xBFFC08FEL	#define VTEX10XXA_ERROR_TRIG_TIMER_OUT_OF_RANGE	Trigger timer out of range.
0xBFFC08FFL	#define VTEX10XXA_ERROR_TRIG_COUNT_OUT_OF_RANGE	Trigger count out of range.
0xBFFC0900L	#define VTEX10XXA_ERROR_TRIG_DELAY_OUT_OF_RANGE	Trigger delay out of range.
0xBFFC0901L	#define VTEX10XXA_ERROR_ARM_DELAY_OUT_OF_RANGE	Arm delay out of range.
0xBFFC0902L	#define VTEX10XXA_ERROR_INVALID_TRIG_INF_VALUE	Invalid trigger infinite value.
0xBFFC0903L	#define VTEX10XXA_ERROR_INVALID_ARM_INF_VALUE	Invalid arm infinite value.
0xBFFC0904L	#define VTEX10XXA_ERROR_INVALID_NUM_SCANS	Invalid number of scans.
0xBFFC0905L	#define VTEX10XXA_ERROR_READ_FIFO_TIMEOUT	Timeout on read FIFO.
0xBFFC0906L	#define VTEX10XXA_ERROR_STRMING_DATA_ENABLED	Streaming data already enabled.
0xBFFC0907L	#define VTEX10XXA_ERROR_ENABLING_STRMING_DATA	Enabling streaming data failed.
0xBFFC0908L	#define VTEX10XXA_ERROR_STRMING_DATA_NOT_ENABLED	Streaming data not enabled.
0xBFFC0909L	#define VTEX10XXA_ERROR_DISABLING_STRMING_DATA	Disabling streaming data failed.
0xBFFC090AL	#define VTEX10XXA_ERROR_INSUFFICIENT_FIRMWARE_REV	Insufficient firmware revision.
0xBFFC090BL	#define VTEX10XXA_ERROR_VXI_11	VXI-11 Error.
0xBFFC090CL	#define VTEX10XXA_ERROR_INVALID_FILTER_FREQ	Invalid filter frequency.
0xBFFC090DL	#define VTEX10XXA_ERROR_STRMIN_ALREADY_ENABLED	Streaming data already enabled on instrument.
0xBFFC090EL	#define VTEX10XXA_ERROR_INVALID_NUM_ENTRIES	Invalid number of entries.
0xBFFC090FL	#define VTEX10XXA_ERROR_NEGATIVE_TIME	Illegal negative time.
0xBFFC0910L	#define VTEX10XXA_ERROR_DEVICE_LOCKED	Device locked.
0xBFFC0911L	#define VTEX10XXA_ERROR_DEVICE_NOT_LOCKED	Device not locked.
0xBFFC0912L	#define VTEX10XXA_ERROR_CALIBRATION_IN_PROGRESS	Calibration in progress.
0xBFFC0913L	#define VTEX10XXA_ERROR_CALIBRATION_MISSING	Calibration missing.
0xBFFC0914L	#define VTEX10XXA_ERROR_TRIGGER_SYSTEM_ERROR	Trigger system error.
0xBFFC0915L	#define VTEX10XXA_ERROR_INSUFFICIENT_UPTIME	Insufficient uptime for requested operation.
0xBFFC0916L	#define VTEX10XXA_ERROR_OUT_OF_LINK_IDS	Maximum number of link IDs exceeded.

Error Code	Error Message	Meaning
0xBFFC0917L	#define VTEX10XXA_ERROR_INVALID_LINK_ID	Invalid link ID.
0xBFFC0918L	#define VTEX10XXA_ERROR_DUPLICATE_SCANLIST_CHNL	Duplicate channel In scanlist.
0xBFFC0919L	#define VTEX10XXA_ERROR_SELF_CAL_MISSING	Self-calibration missing.
0xBFFC091AL	#define VTEX10XXA_ERROR_UNKNOWN_FIRMWARE_ERROR	Unknown firmware error.
0xBFFC091BL	#define VTEX10XXA_ERROR_INVALID_DIO_LIMIT	Invalid DIO limit value.
0xBFFC091CL	#define VTEX10XXA_BOOLEAN_OUTA_RANGE	Out of range Boolean value.
0xBFFC091DL	#define VTEX10XXA_NOT_A_VALID_RSRC	Not a valid resource string.
0xBFFC091EL	#define VTEX10XXA_ERROR_ALREADY_LOCKED_U	Instrument already locked by this host.
0xBFFC091FL	#define VTEX10XXA_ERROR_ALREADY_LOCKED_NU	Instrument already locked by another host.
0xBFFC0920L	#define VTEX10XXA_ERROR_ALREADY_UNLOCKED	Instrument is not locked at this time.
0xBFFC0921L	#define VTEX10XXA_ERROR_NONVOL_SELF_CAL_MISSING	Nonvolatile self calibration data missing.
0xBFFC0922L	#define VTEX10XXA_ERROR_FIFO_EMPTY	FIFO is empty.
0xBFFC0923L	#define VTEX10XXA_ERROR_STRING_TOO_BIG	Input buffer contains too many characters.
0xBFFC0924L	#define VTEX10XXA_ERROR_PTP_NOT_RUNNING	Precision time protocol not running. Try reconfiguring time settings, or a power cycle.
0xBFFC0925L	#define VTEX10XXA_ERROR_INVALID_BOOLEAN_VALUE	Invalid Boolean value.
0xBFFC0926L	#define VTEX10XXA_ERROR_WIREDOR_MUST_BE_SET	Wired-OR must be set.
0xBFFC0927L	#define VTEX10XXA_ERROR_INVALID_TIMER_COUNT	Invalid timer count.
0xBFFC0928L	#define VTEX10XXA_ERROR_INVALID_TIMER_TIME	Invalid time setting.
0xBFFC0929L	#define VTEX10XXA_ERROR_INVALID_LAN_SOURCE	Invalid LAN source.
0xBFFC092AL	#define VTEX10XXA_ERROR_INVALID_LAN_EVENT_DOMAIN	Invalid LAN domain.
0xBFFC092BL	#define VTEX10XXA_ERROR_INVALID_LAN_EVENT_EVENTID	Invalid LAN Event ID.
0xBFFC092CL	#define VTEX10XXA_ERROR_INVALID_LAN_EVENT_FILTER	Invalid LAN filter.
0xBFFC092DL	#define VTEX10XXA_ERROR_INVALID_OP_WHILE_LOG_ENABLED	This operation is not valid while LAN event logging is enabled.
0xBFFC092EL	#define VTEX10XXA_ERROR_PPS_REQUIRES_DIO0_OUTPUT_LOW	PPS enable requires DIO0 output to be enabled and low.
0xBFFC092FL	#define VTEX10XXA_ERROR_GAIN_OUT_OF_RANGE	Linear correction gain value out of range.
0xBFFC0930L	#define VTEX10XXA_ERROR_OFFSET_OUT_OF_RANGE	Linear correction offset value out of range.
0xBFFC0931L	#define VTEX10XXA_ERROR_INVALID_CORRECTION_LIST_LENGTH	Linear correction list length must be the same as channel list length.
0xBFFC0932L	#define VTEX10XXA_ERROR_INVALID_GAIN_LIST_LENGTH	Gain list length must be the same as channel list length.
0xBFFC0933L	#define VTEX10XXA_ERROR_OPERATION_VALID_ONLY_ON_VOLT_CH	This operation is only valid on general purpose voltage channels.
0xBFFC0934L	#define VTEX10XXA_ERROR_INVALID_EVENTLOG_OVERFLOWMODE	Invalid event log overflow mode.
0xBFFC0935L	#define VTEX10XXA_ERROR_INVALID_CAL_FILE_TYPE	Invalid calibration file type.
0xBFFC0936L	#define VTEX10XXA_ERROR_ACQUISITION_IN_PROGRESS	Acquisition in progress.
0xBFFC0937L	#define VTEX10XXA_ERROR_FACTORY_MODE_ONLY	Only works when **calFileType** parameter is a value greater than 0.
0xBFFC0938L	#define VTEX10XXA_ERROR_CAL_FILE_TOO_LARGE	Unable to load the entire calibration file into memory.
0xBFFC0939L	#define VTEX10XXA_ERROR_CAL_MODE_ONLY	The calibration mode must be > 0.
0xBFFC093AL	#define EX1048_ERROR_NO_SERIAL_NUMBER	A device without a serial number has made a call has that requires a serial number.
0xBFFC093BL	#define EX1048_ERROR_NO_MAC_ADDR	A device without a MAC address has made a call has that requires a MAC address.
0xBFFC093CL	#define EX1048_ERROR_CAL_FILENAME_TOO_LONG	The calibration file name has overflowed the **calibration[]** buffer set by vtex10xxA_get_calibration_file.
0xBFFC093DL	#define EX1048_ERROR_LOADING_CAL_FILE	The device is unable to load the calibration file (i.e., the file may not exist).
0xBFFC093EL	#define EX1048_ERROR_INVALID_VOLTAGE_RANGE	A channel has been set to an invalid voltage range (i.e. a thermocouple channel has been set to 10 V while in voltage mode).
0xBFFC093FL	#define EX1048_ERROR_INVALID_RANGE_FOR_TC_CONVERSION	TC conversions can only be in the 67 mV range.
0xBFFC0940L	#define EX1048_ERROR_OTD_NOT_ALLOWED_IN_10V_RANGE	Hardware is not useful on this range.

SECTION 8

THEORY OF OPERATION

INTRODUCTION

The block diagram in Figure 6-1 illustrates the key components of the analog circuitry. Each of the main blocks is described below.

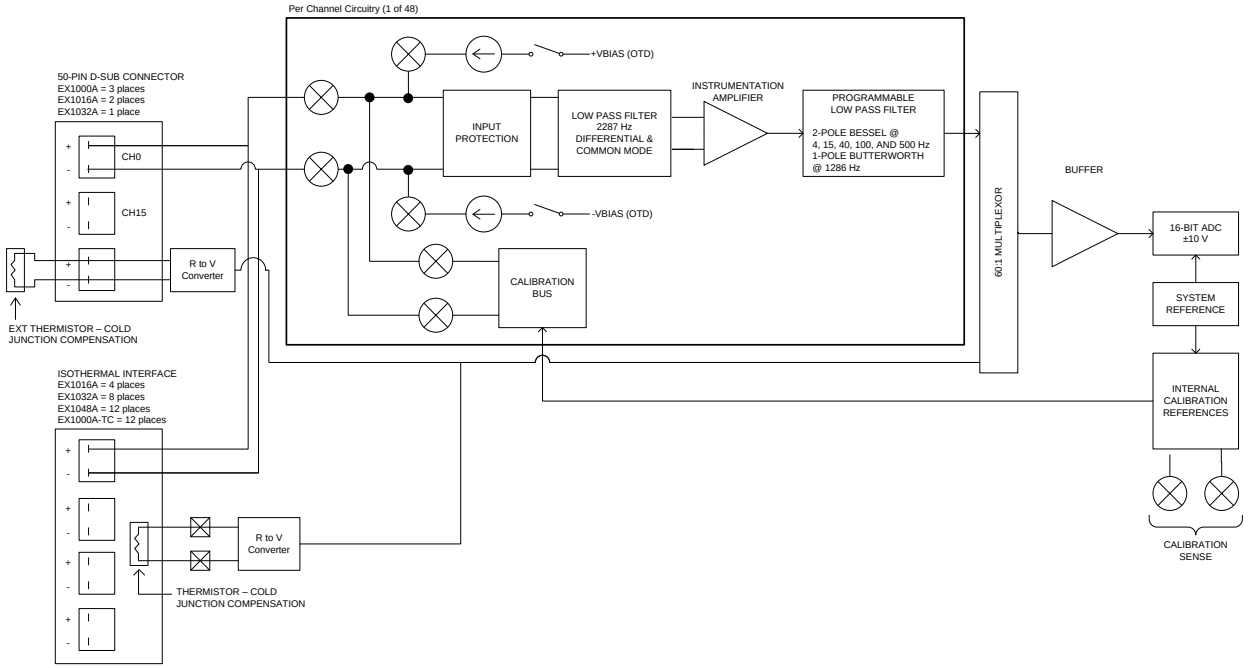


FIGURE 6-1: EX10XXA ANALOG CIRCUITRY BLOCK DIAGRAM

VOLTAGE / THERMOCOUPLE INPUT OPTIONS

Each input channel, dependant on the model, will be either a type "U" (copper) miniature thermocouple connector or will be part of a 50-pin D-sub connector. See Table 2-3 for pin assignments of the D-sub connectors.

Thermocouple Connector Inputs

Each group of four thermocouple connectors is tightly coupled to an internal temperature circuit, providing very accurate cold junction compensation. On either side of the front panel, two nickel plated banana jacks as available which connect directly to the chassis. These jacks allow the shield to be connected to chassis using short wires when shielded thermocouples or test wires are used.

D-sub Connector Inputs

Each D-sub connector provides sixteen input channels which are grouped around the perimeter of the connector (see Table 2-3). Two pins on these connectors are allocated to the connection of an thermistor, typically used with an external cold plate. The center row of each D-sub is connected to chassis. When using shielded test wires or thermocouples, these pins allow the shield to be connected to chassis using very short wires.

SIGNAL CONDITIONING CIRCUITRY

Each input channel is conditioned by the block labeled “Per Channel Circuitry” in Figure 6-1. The input protection circuitry ensures that only a safe level of current will flow during over-voltage conditions.

Open transducer detection (OTD) is provided by nanoamp-level current sources (~5 nA to 8 nA) on each input path. These current sources are small enough not to affect measurement accuracy, but large enough to drive the high-impedance input of the instrumentation amplifier (IA) deterministically into saturation in the event of an open condition. When enabled, they provide continuous monitoring of the input and will generate an open indication event when the open is intermittent in nature. Moreover, these current sources are provided on both inputs so that an open condition is registered even in the case where only one lead of the input is open and the other is connected to earth ground. An important feature of the OTD is that the current sources can be enabled or disabled on a per channel basis. This permits the user to choose which of their input signals require open circuit detection.

The core of this block is the IA that provides amplification of the differential input signal and rejection of common mode input signals. The IA provides excellent CMRR (common mode rejection ratio), especially at dc and (50/60) Hz, but its rejection characteristic decreases as the interference frequency increases. Moreover, IAs have the tendency to shift their dc offset in the presence of very high frequency signals. This effect would be particularly problematic, as the filters that follow the IA would not attenuate it. However, the presence of the differential and common mode filter preceding the IA attenuates high frequency interference before it reaches the IA. This decreases the possibility of dc rectification and provides the system with excellent CMRR characteristics at all frequencies.

For optimum noise performance, the IA output is routed through one of six user selectable filters. Five of the six filters are two-pole, Bessel response (constant delay) with cutoff frequencies of 4 Hz, 15 Hz, 40 Hz, 100 Hz and 500 Hz. The sixth filter is a single pole, Butterworth response, with a 1000 Hz cutoff frequency. Each channel may have any one of the six filters applied to its signal path. These filters form a powerful application tool, as it allows the bandwidth of the signal conditioning path to be matched to the characteristics of the attached sensor. Of particular note, the filters used are continuously connected.

Contrasting approaches utilize one filter whose characteristics are modified based on user selection. The disadvantage to that approach is that there is an inherent latency when the filter changes state before the data is valid. This requires the user to impose a potentially long delay in their measurement system. The per channel filter approach used in the EX10xxA has no such characteristic and requires no delay. Figure 6-2 shows the typical CMRR data for the EX10xxA at each filter setting (4 Hz, 15 Hz, 40 Hz, 100 Hz, 500 Hz, to 1000 Hz) when the common mode input is 20.118 V p-p, 60 Hz.

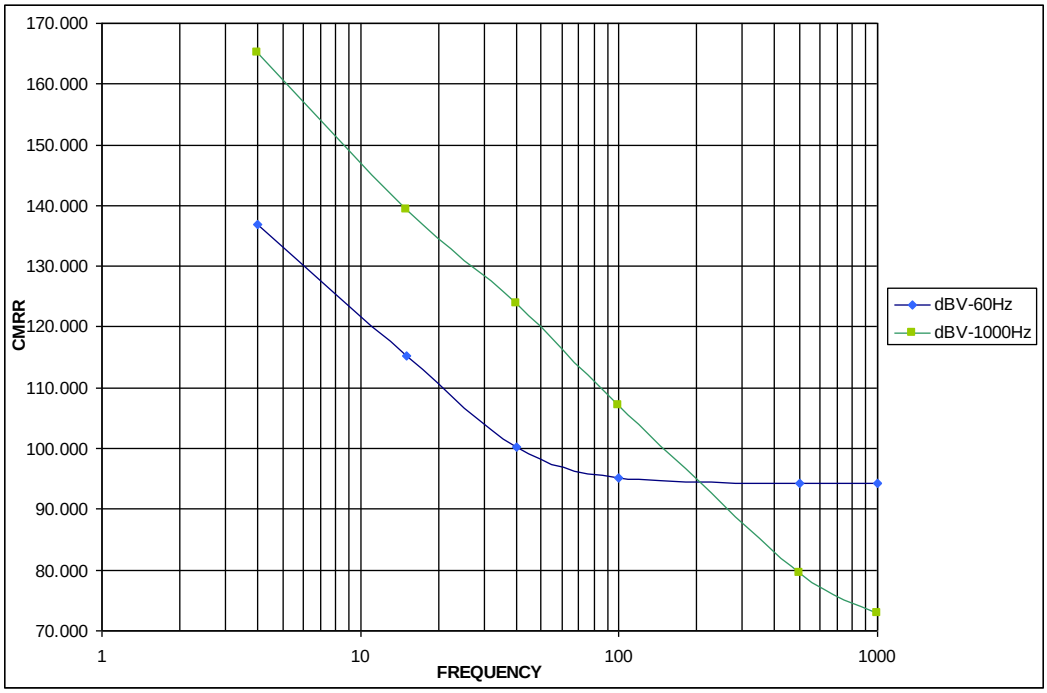


FIGURE 6-2: TYPICAL CMRR AT 60 HZ AND 1 KHZ

COLD JUNCTION COMPENSATION

Cold junction compensation is accomplished via twelve precision thermistors that monitor isothermal interface on the four input connectors. A thermistor is an element whose resistance represents its applied temperature. Each thermistor resistance is converted to a representative voltage that is measured at the beginning of each scan. Each voltage is then transformed into a temperature value through the application of the thermistor's Steinhart-Hart coefficients. This temperature value is then used to generate a thermocouple type-specific compensating voltage that is mathematically added to the measured input voltage of the appropriate channels.

CALIBRATION

Calibration in the EX10xxA takes two forms: full calibration and self-calibration. In both cases, the input signal conditioning paths are disconnected from the input jacks and connected instead to a calibration bus that is driven by an internal calibration source. Through measurement of the conditioning paths at multiple calibration source points, the voltage gain and offset of each path is calculated. Full calibration involves the additional steps of measuring the calibration source with a precision voltmeter. Thus, self-calibration is a subset of full calibration. Because of the internal input disconnection mechanism, the user does not have to remove the actual input connections to perform calibration.

THERMOCOUPLE CALCULATIONS

There are two thermocouple type-specific calculations that are performed in the EX10xxA. The first calculation transforms the CJC temperature into a compensating voltage that is mathematically added to the measured input voltage. The second calculation transforms this total voltage into its final thermocouple temperature. For maximum accuracy, both of these calculations are performed using the full-order polynomial equations and coefficients from the NIST ITS-90 Thermocouple Database, not from lookup tables or piecewise linear approximations.

SECTION 9

EX10SC SIGNAL CONDITIONING

OVERVIEW

The EX10SC modular signal conditioning chassis is designed EX10xxA instruments. This modular chassis allows users to add signal conditioning capabilities not currently addressed by the EX10xxA family.

Compatibility Table				
	EX1000A	**EX1016A**	**EX1032A**	**EX1048A**
EX10SC	**Yes**	**Yes**	**Yes**	**No**

TABLE 9-1: EX10XXA FAMILY COMPATIBILITY WITH EX10SC

The EX10SC is designed to be easy-to-implement and is designed with groups of sixteen channels. The additional signal conditioning capability comes through the use of 8B signal conditioning plug-ins. One 8B module is provided per channel. The EX10SC is intended for users that required channel-to-GND and channel-to-channel isolation.

Additional capabilities addressed by the EX10SC are:

- Wider range of input voltages ±60 V
- Isolation up to 300 V (input to ground), 1500 V rms isolation (module only)for industrial and voltage stack measurements (i.e. solar cells and batteries)
- 4 mA to 20 mA current measurement loop capability
- Three- and four-wire RTD measurement capability
- Full-bridge strain gage/bridge measurements
- Frequency measurement capability

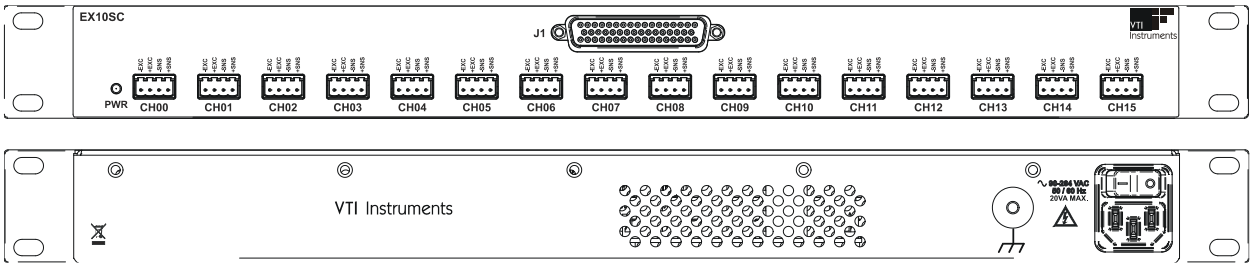


FIGURE 9-1: EX10SC (FRONT AND REAR PANELS)

CONNECTING POWER TO THE EX10SC

Connect mains power to the EX10SC power receptacle located at the rear of the mainframe. Prior to powering on the EX10SC, ensure that all connections have been made, that proper grounding has been observed, and that mains power does not exceed the power rating noted in the *EX10SC General Specifications* section that follows.

NOTES	It is important to note that, if connected to a high energy source, damage to the EX10xxA and/or EX10SC systems and the plug-ins installed can occur. To avoid damage due to high energy sources, the following precautions should be made: 1) Limit the input current and/or power to the values listed in the plug-ins maximum specifications. 2) Employ current/power limiting devices (such as fuses) to ensure that excessive current/power is mitigated in the event a system fault occurs.

CONNECTING TO AN EX10XXA TO THE EX10SC

The EX10SC output signals are designed to be routed directly to the input channels of the EX1000A, EX1016A or EX1032A via a D-sub cable. The cable (VTI P/N 70-0397-000) directly connects the EX10SC outputs to the user selected input D-sub connector on either the EX1000A, EX1016A, or EX1032A.

Connector Pin/Signal Assignment

The table below provide signal and connector pin information for the interconnect cable. For mating connector information, please refer to the *Accessories* section of the *EX10SC General Specifications*.

Pin	Signal	Pin	Signal	Pin	Signal
1	CJC_RETURN	**18**	CHASSIS GND	**34**	CJC_INPUT0
2	+OUTPUT_CH7	**19**	CHASSIS GND	**35**	-OUTPUT_CH15
3	-OUTPUT_CH7	**20**	CHASSIS GND	**36**	+OUTPUT_CH15
4	+OUTPUT_CH6	**21**	CHASSIS GND	**37**	-OUTPUT_CH14
5	-OUTPUT_CH6	**22**	CHASSIS GND	**38**	+OUTPUT_CH14
6	+OUTPUT_CH5	**23**	CHASSIS GND	**39**	-OUTPUT_CH13
7	-OUTPUT_CH5	**24**	CHASSIS GND	**40**	+OUTPUT_CH13
8	+OUTPUT_CH4	**25**	CHASSIS GND	**41**	-OUTPUT_CH12
9	-OUTPUT_CH4	**26**	CHASSIS GND	**42**	+OUTPUT_CH12
10	+OUTPUT_CH3	**27**	CHASSIS GND	**43**	-OUTPUT_CH11
11	-OUTPUT_CH3	**28**	CHASSIS GND	**44**	+OUTPUT_CH11
12	+OUTPUT_CH2	**29**	CHASSIS GND	**45**	-OUTPUT_CH10
13	-OUTPUT_CH2	**30**	CHASSIS GND	**46**	+OUTPUT_CH10
14	+OUTPUT_CH1	**31**	CHASSIS GND	**47**	-OUTPUT_CH9
15	-OUTPUT_CH1	**32**	CHASSIS GND	**48**	+OUTPUT_CH9
16	+OUTPUT_CH0	**33**	CHASSIS GND	**49**	-OUTPUT_CH8
17	-OUTPUT_CH0			**50**	+OUTPUT_CH8

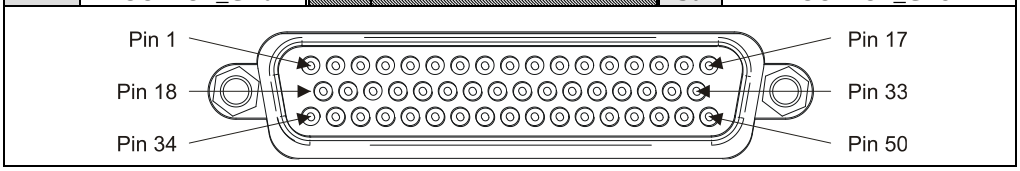


TABLE 9-2: EX10SC INTERCONNECT CABLE CONNECTOR PIN SIGNAL ASSIGNMENT

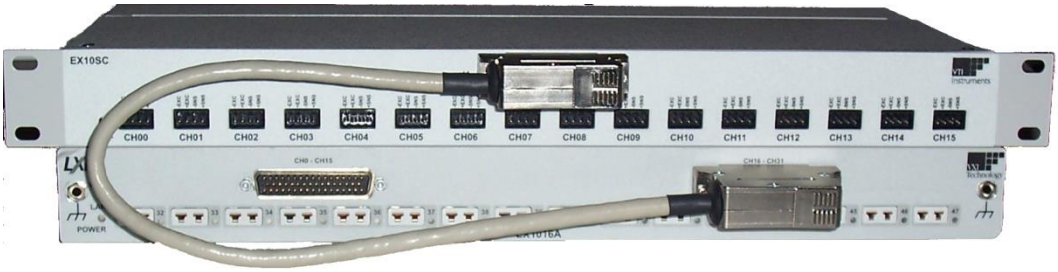


FIGURE 9-2: EX1016A CONNECTED TO THE EX10SC

The cable routes the channels in the following manner:

D-Sub Cable	To EX1000A	To EX1016A	To EX1032A	Notes
From EX10SC	CH0-CH15			CH00 wires to CH 0
From EX10SC	CH16-CH31			CH00 wires to CH 16
From EX10SC	CH32-CH47			CH00 wires to CH 32
From EX10SC		CH0-CH15		CH00 wires to CH 0
From EX10SC		CH16-CH31		CH00 wires to CH 16
From EX10SC			CH32-CH47	CH00 wires to CH 32

FIGURE 9-3: EX10XXA AND EX10SC CHANNEL CONNECTIONS

CONNECTING TO THE EX10SC TO THE DUT

EX10SC is supplied with sixteen connectors (VTI P/N 27-0400-004) for connecting to the device under test (DUT). This allows the user to connect all field wiring before connecting the DUT to the EX10SC input channels. Each input channel on the EX10S is labeled **-EXC**, **+EXC**, **-SNS**, and **+SNS**.

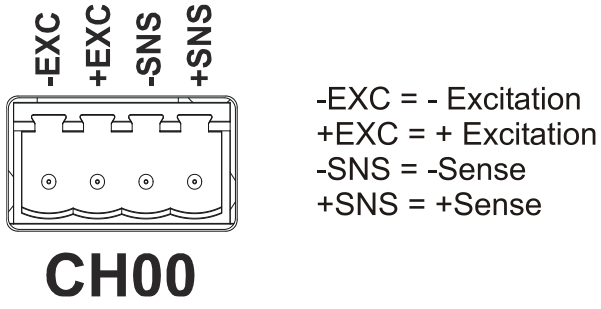


FIGURE 9-4: EX10SC INPUT CONNECTOR

Since the 8B modules require only power and no control, no analog integration with the EX10xxA is required. The 0 V to 5 V and +5 V to -5 V output modules were selected to match the front panel capabilities of the EX1000A, EX1016A, and EX1032A. No hardware modifications to the EX10xxA family of products should be required, however software driver modifications to support additional signal types may. Because of the modular nature of the 8B modules, other modules may be added if the business case dictates this.

For most 8B modules, calibration is not required. For the thermocouple type modules, the user can compensate for error at the 0 °C reading by doing the following:

1) Using a thermocouple calibrator, connect the leads to connect to the **+SNS** and **-SNS** leads inputs of the EX10SC.
2) Set the calibrator to 0 °C and determine the temperature.
3) Enter the module's offset, in °C, the application software (such as EXLab).

CHANGING SIGNAL CONDITIONING MODULES

The EX10SC is designed to allow the customer to change the 8B modules if test requirements change or when a module becomes non-functional. Before removing an 8B module, ensure that the mainframe has been powered down and that the power cable has been removed from the unit. Proper ESD precautions should be taken when changing a signal conditioning unit and removal should only be performed by a qualified technician.

The following process can be used to remove an 8B module:

1) Remove all power from the EX10SC mainframe.
2) Using a #2 Phillips screwdriver, loosen the quarter-turn captive screws located at the top of the EX10SC near the front of the chassis. These screws will remain attached to the lid once they are loosened.
3) Next, find the 8B module that will be removed and, using the #2 Phillips screw driver, loosen the screw until it is no longer secured in the PCB.
4) Remove the 8B module from the captive jacks on the PCB.
5) Next, insert the replacement 8B module onto the PCB, securing the module with the captive Phillips screw on the module.
6) Close the lid of the EX10SC and tighten the quarter-turn captive screws.

This feature allows the user flexibility when using the EX10SC, as a variety of 8B modules can be installed in the same mainframe and can also be mixed-and-matched after the unit has been fielded. In addition to changing the 8B module, jumper and shunt resistor requirements must also be considered. The location of the 8B module, the jumpers, and the shunt resistor sockets are shown in Figure 9-6.

Jumper Settings

For most 8B modules, jumpers must be installed to connect the module to an on-board thermistor. The thermistor is used for cold junction compensation on thermocouple modules, while is used for temperature compensation on others. Table 9-3 shows which modules require connection of the JP*x* jumper on the EX10SC PCB.

Module	Description	Jumper JP*x**	Resistor
8B32-02	20 mA current input	CONNECTED	OPTIONAL
8B42-01	4 mA to 20 mA 2-wire transmitter	CONNECTED	OPTIONAL
8B33-03	0 V to 10 V RMS voltage	CONNECTED	UNUSED
8B33-04	0 V to 100 V RMS voltage	CONNECTED	UNUSED
8B33-05	0 V to 300 V RMS voltage	CONNECTED	UNUSED
8B34-04	2-/3-wire RTD (100 Ω Pt, 0 °C to 600 °C)	CONNECTED	UNUSED
8B36-04	0 Ω to 10 kΩ potentiometer	CONNECTED	UNUSED
8B41-01	±1 V voltage input	CONNECTED	UNUSED
8B41-03	±10 V voltage input	CONNECTED	UNUSED
8B41-07	±20 V voltage input	CONNECTED	UNUSED
8B41-09	±40 V voltage input	CONNECTED	UNUSED
8B41-12	±60 V voltage input	CONNECTED	UNUSED
8B45-02	0 Hz to 1 kHz frequency input	CONNECTED	UNUSED
8B45-05	0 Hz to 10 kHz frequency input	CONNECTED	UNUSED
8B45-08	0 Hz to 100 kHz frequency input	CONNECTED	UNUSED
8B47J-12	Type J thermocouple	CONNECTED	UNUSED
8B47K-13	Type K thermocouple	CONNECTED	UNUSED
8B47T-06	Type T thermocouple	CONNECTED	UNUSED
8B35-04	4-wire RTD (100 Ω Pt, 0 °C to 600 °C)	UNUSED	UNUSED
8B38-01	±10 mV strain gage	UNUSED	UNUSED
8B38-02	±30 mV strain gage	UNUSED	UNUSED

TABLE 9-3: 8B MODULE JUMPER SETTINGS

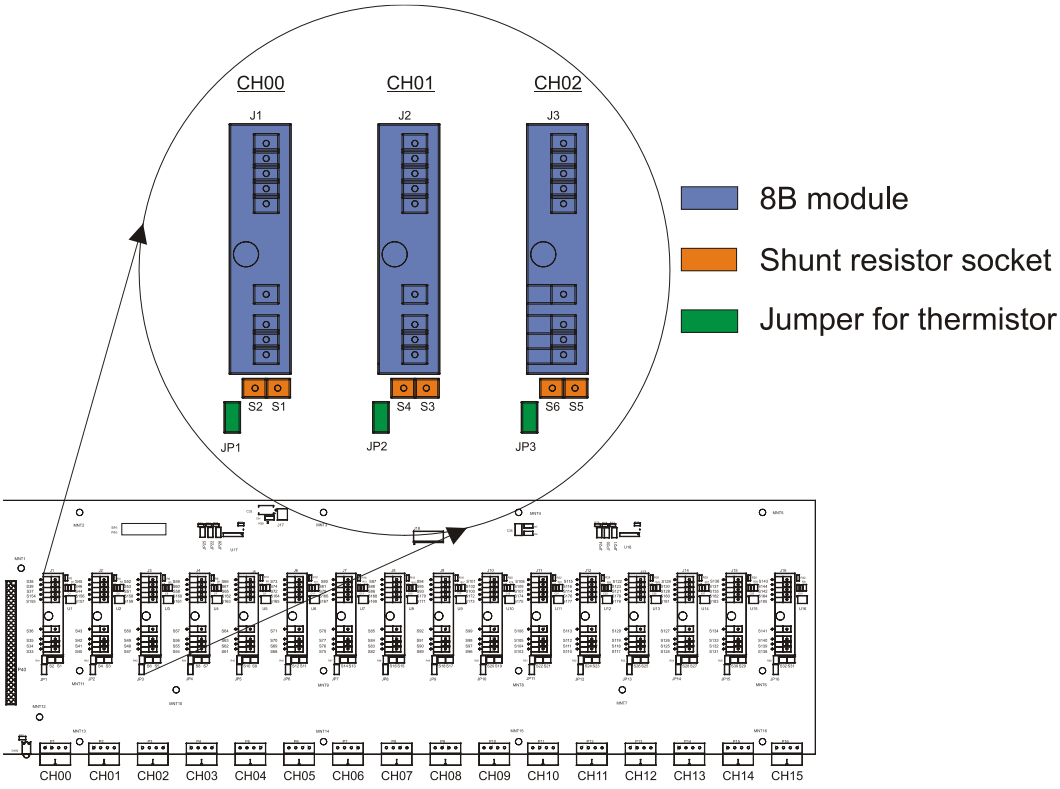


FIGURE 9-5: COMPONENT LOCATIONS ON EX10SC PCB

Shunt Resistors

Optional shunt resistors can be installed by the user into sockets provided on the PCB. These resistors allow for larger currents to be monitored when using the 8B32-02 or 8B42-01 modules. The shunt resistor sockets are spaced on a 0.150" center. The sockets accept Vishay series S102 resistors, but similar resistors from other manufacturers can be used as well. Table 9-2 below shows the relationship between the channel and its associated sockets.

Channel	Jumper	Shunt Resistor Sockets
CH00	J1	S1 and S2
CH01	J2	S3 and S4
CH02	J3	S5 and S6
CH03	J4	S7 and S8
CH04	J5	S9 and S10
CH05	J6	S11 and S12
CH06	J7	S13 and S14
CH07	J8	S15 and S16
CH08	J9	S17 and S18
CH09	J10	S19 and S20
CH10	J11	S21 and S22
CH11	J12	S23 and S24
CH12	J13	S25 and S26
CH13	J14	S27 and S28
CH14	J15	S29 and S30
CH15	J16	S31 and S32

FIGURE 9-6: CHANNEL, JUMPER, AND SOCKET MAPPING

SOFTWARE INTERFACE

As a signal conditioning device, the EX10SC is a passive device and adds no additional APIs to the EX10xxA. All you have to do is to scale data based on the SC module connected to the channel. The signal conditioners function similar to a transducer and, based on the 8B module being used, scale factors can be used from the 8B module's data sheet to adjust the readings.

When EXLab is used, the scale factors are accounted for using its GUI interface, making the additional calculations unnecessary. The steps below of using EXLab with and EX10xxA and EX10SC is provided below as an example.

7) Add the EX10xxA in Agilent Connection Expert (ACE). Refer to the software's user's manual or help file for instructions. Note that an EX1016A is being used in the example and that the EX10SC does not appear.

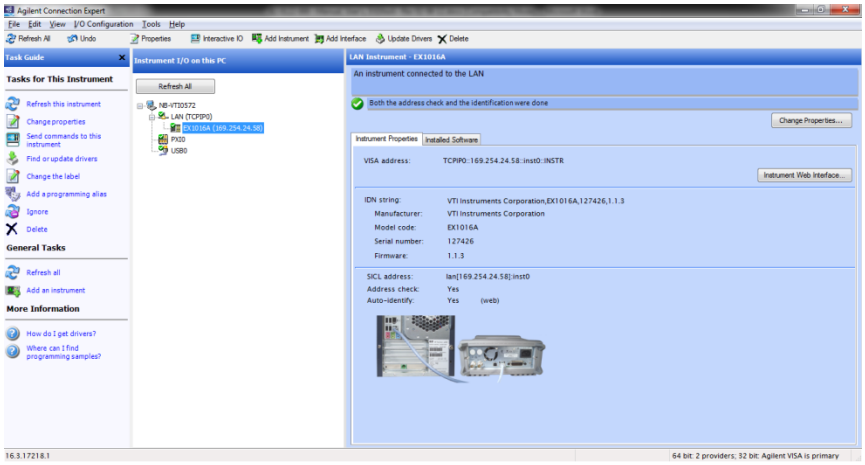


FIGURE 9-7: ADDING AN EX1016A TO ACE

8) Once the EX1016A is added to ACE, EXLab can be used to view the instrument. Simply click on **Tools->Instrument Discovery** under the menu bar. The **Device Search** wizard will be shown. Click on **Search for Devices,** check the EX1016A and click OK.

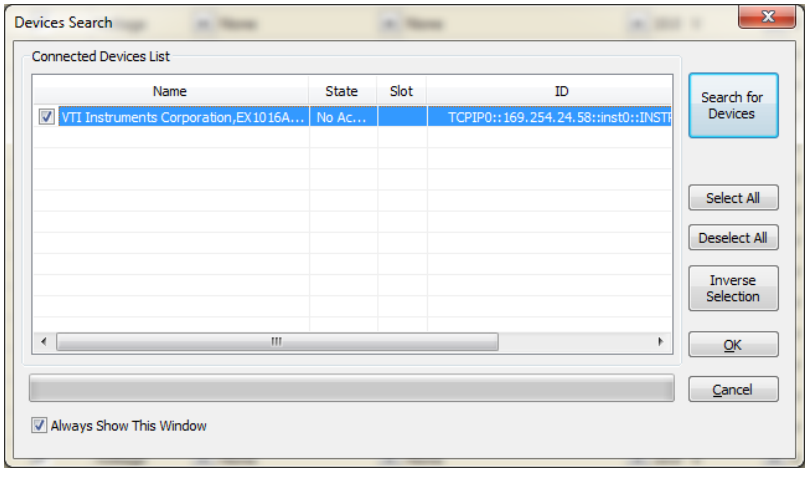


FIGURE 9-8: ADDING AN EX1016A TO EXLAB

9) The **Sig Conditioner** column is used to specify the signal conditioner used in a specific channel. Note that, in the drop menu, many EX10SC 8B modules are predefined and can be selected.

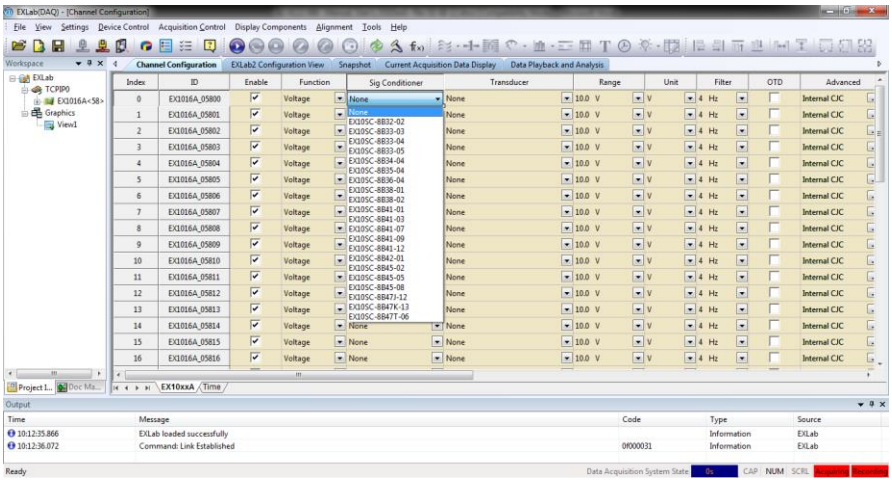


FIGURE 9-9: SELECTION OF A PREDEFINED 8B MODULE

10) If the required 8B module is not listed (or if another type of signal conditioning module is used), a custom signal conditioner can be defined. To create a custom signal conditioner, simply click **Tools->Transducer Library** under the menu bar. The **Transducer Library and Signal Conditioner** wizard will be shown. Highlight the type of the transducer in the section on the left and click on **Add.** The **Add Transducer** wizard will be shown and it allows the user to enter the information of the transducer.

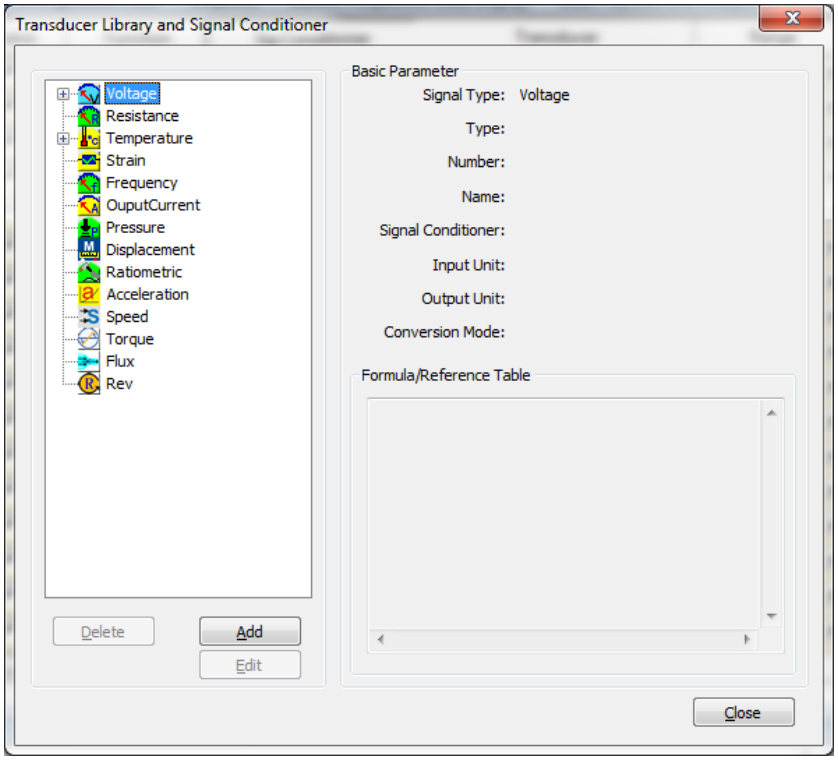


FIGURE 9-10: TRANSDUCER LIBRARY AND SIGNAL CONDITIONER WIZARD

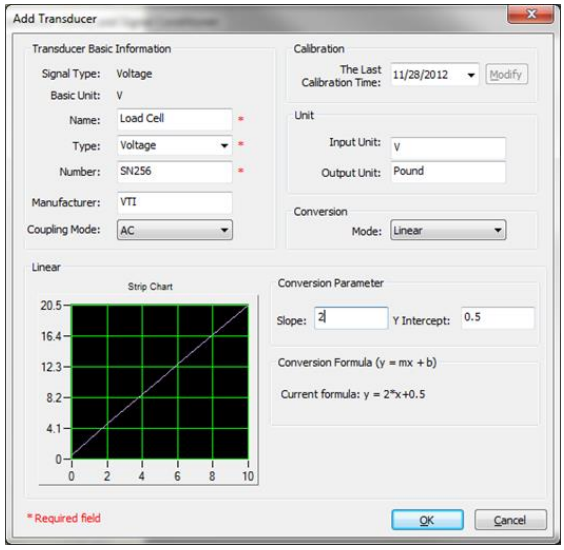


FIGURE 9-11: ADD TRANSDUCER WIZARD

- **Name** – The name of the new transducer.
- **Type** – The type of the new transducer. The user can type in the value or choose in the drop down list by clicking the down arrow in the field.
- **Number** – This number is important and will be shown up as a new in the channel configuration page while the user selects to use on channels.
- **Manufacturer** – The manufacturer of the transducer.
- Coupling Mode – The coupling more of the transducer.
- **Conversion Mode** – This supports Linear, Polynomial, Lookup Table and Division Table
- **The Last Calibration Time** – The time of the last calibration.
- **Input Unit** – The unit of the input signal going into the transducer.
- **Output Unit** – The unit of the output signal going out from the transducer.

11) Once the signal conditioner is defined, click OK and the new transducer should be shown in the **Transducer Library and Signal Conditioner** wizard.

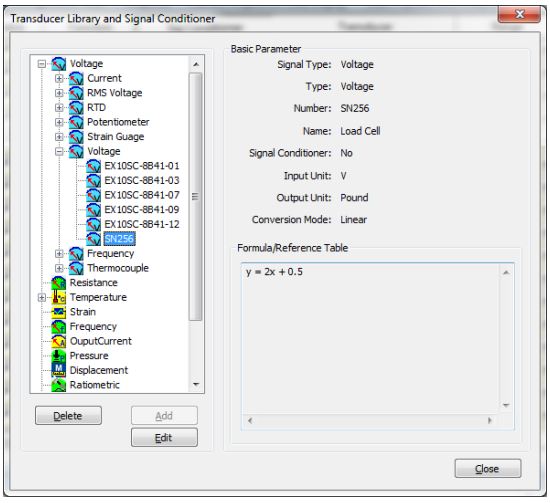


FIGURE 9-12: NEW TRANSDUCER SHOWN IN THE TRANSDUCER LIBRARY AND SIGNAL CONDITIONER WIZARD

12) After the transducer has been successfully added, the new transducer can be selected in the **Transducer** column.

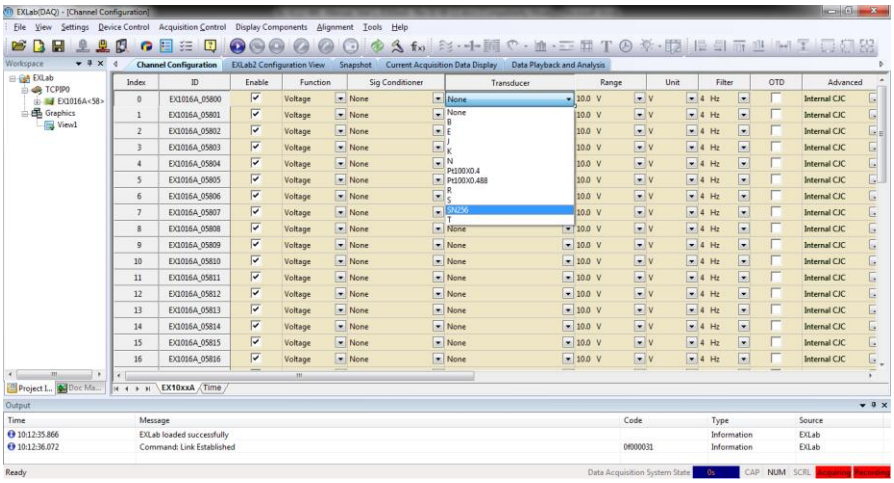


FIGURE 9-13: NEW TRANSDUCER SHOWN UP IN THE TRANSDUCER COLUMN

For more information on configuring and using EXLab to display and record data, please refer to the EXLab user manual provided with the software.

EX10SC General Specifications

EX10SC Specifications	
Number of Channels	
	16
Dimensions	
	1.75” H x 17.94” W x 7.75” D
Input Power	
Input voltage/frequency	90 V ac – 264 V ac*, 50 Hz/60 Hz (nominal ac)
Power Consumption	
Without modules	3 VA
Fully populated with modules	30 VA maximum
Accessories	
Cable	
Description	Cable, EX10SC-CBL01, EX10SC to EX10xxA
VTI part number	70-0397-000
Mating Connectors	
Description	Connector, terminal block, screw, 4-position (supplied with EX10SC)
VTI part number	27-0400-004
Mfg part number	RIA Connect 31169104
Screwdriver	
Description	Screwdriver for use with 27-0400-004
Mfg part number	RIA Connect 791989
Tabletop Kit	
Description	Table top kit for EX10XXA series
VTI part number	70-0355-902
Rackmount Options	
Description	Rackmount kit for EX10xxA series
VTI part number	70-0355-900
Description	Rackmount slide rails
VTI part number	EX10SC-RK001

EX10SC MODULE SPECIFICATIONS

EX10SC-8B32-02 SPECIFICATIONS	
DESCRIPTION	
	0 mA to 20 mA input
INPUT RANGE	
	0 mA to 20 mA or 4 mA to 20 mA
INPUT RESISTANCE	
Normal	<50 Ω
Power Off	<50 Ω
INPUT PROTECTION	
Continuous	40 V ac
Transient	ANSI/IEEE C37.90.1
CMV (COMMON MODE VOLTAGE)	
Input to output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50Hz or 60Hz	120 dB
NMR (NORMAL MODE REJECTION)	
	70 dB at 60 Hz
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±25 ppm/°C
Gain	±50 ppm/°C
NOISE	
Output	100 kHz, 250 µV rms
BANDWIDTH (-3 dB)	
	3 Hz
RESPONSE TIME	
90% Span	150 ms
MTBF	
	584,000 hr

EX10SC-8B34-04 SPECIFICATIONS	
DESCRIPTION	
	2- and 3-wire 100 Ω RTD (0 °C to 600 °C)
INPUT RANGE LIMITS	
Input Range	0 °C t o +600 °C (+32 °F to +1112 °F)
Accuracy	±0.45 °C
INPUT RESISTANCE	
Normal	50 MΩ
Power Off	200 kΩ
Overload	200 kΩ
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
SENSOR EXCITATION CURRENT	
	0.25 mA
LEAD RESISTANCE EFFECT	
	±0.02 °C/Ω
CMV (COMMON MODE VOLTAGE)	
Input to Output	1500 V rms maximum
TRANSIENT	
Input to Output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	120 dB
NMR (NORMAL MODE REJECTION)	
	70 dB at 60 Hz
ACCURACY	
	See ordering information
STABILITY	
Offset	±20 ppm/°C
Gain	±50 ppm/°C
NOISE	
Output, 100 kHz	200 µV rms
BANDWIDTH (-3 dB)	
	3 Hz
RESPONSE TIME	
90% span	150 ms
RTD STANDARDS	
100 Ω Pt	
Alpha Coefficient	0.00385
DIN	DIN 43760
JIS	JIS C 1604-1989
IEC	IEC 751

EX10SC-8B36-04 SPECIFICATIONS	
DESCRIPTION	
	Potentiometer input (0 Ω to 10 kΩ)
INPUT RANGE	
	0 to 10 kΩ
INPUT RESISTANCE	
Normal	50 MΩ
Power off	200 kΩ
Overload	200 kΩ
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
SENSOR EXCITATION CURRENT	
100 Ω, 500 Ω, 1 kΩ sensor	0.25 mA
10 kΩ sensor	0.10 mA
LEAD RESISTANCE EFFECT	
100 Ω, 500 Ω, 1 kΩ sensor	±0.01 Ω/Ω
10 kΩ sensor	±0.02 Ω/Ω
CMV (COMMON MODE VOLTAGE)	
Input to output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	120 dB
NMR (NORMAL MODE REJECTION)	
	70 dB at 60 Hz
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±20 ppm/°C
Gain	±50ppm/°C
NOISE	
Output, 100 kHz	200 µV rms
BANDWIDTH (-3 dB)	
	3 Hz
RESPONSE TIME	
90% span	150 ms

EX10SC-8B33-XX SPECIFICATIONS	
DESCRIPTIONS	
EX10SC-8B33-03	0 V to 10 V rms
EX10SC-8B33-04	0 V to 100 V rms
EX10SC-8B33-05	0 V to 300 V rms
FREQUENCY RANGE	
	45 Hz to 1000 Hz (extended range to 10 kHz) Compatible with standard current and potential transformers
ACCURACY	
	±0.25% factory
ISOLATION	
	1500 V rms transformer
INPUT OVERLOAD PROTECTED	
Peak AC and DC	350 V rms maximum
Continuous	2 A rms
TRANSIENT PROTECTION	
	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	120 dB

EX10SC-8B35-04 SPECIFICATIONS	
DESCRIPTION	
	4-wire 100 Ω RTD (0 °C to 600 °C)
INPUT RANGE LIMITS	
	-200 °C to +850 °C (100 Ω Pt)
INPUT RESISTANCE	
Normal	50 MΩ
Power Off	200 kΩ
Overload	200 kΩ
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
SENSOR EXCITATION CURRENT	
	0.25 mA
LEAD RESISTANCE EFFECT	
	±0.005 °C/Ω
CMV (COMMON MODE VOLTAGE)	
Input to output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	120 dB
NMR (NORMAL MODE REJECTION)	
	70 dB at 60 Hz
STABILITY	
Offset	±20 ppm/°C
Gain	±50 ppm/°C
NOISE	
Output, 100 kHz	200 µV rms
BANDWIDTH (-3 dB)	
	3 Hz
RESPONSE TIME	
90% span	150 ms
100 Ω PT	
Input range	0 °C t o +600 °C (+32 °F to +1112 °F)
Accuracy	±0.45 °C
RTD STANDARDS	
100 Ω Pt	
Alpha coefficient	0.00385
DIN	DIN 43760
JIS	JIS C 1604-1989
IEC	IEC 751

EX10SC-8B38-XX SPECIFICATIONS	
DESCRIPTIONS	
EX10SC-8B38-01	Full-bridge strain (3.33 V excitation)
EX10SC-8B38-02	Full-bridge strain (10 V excitation)
INPUT RANGE	
	±10 mV to ±100 mV
INPUT BIAS CURRENT	
	±0.5 nA
INPUT RESISTANCE	
Normal	50 MΩ
Power Off	100 MΩ
Overload	100 MΩ
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
EXCITATION OUTPUT	
x1	+3.333 V ± 2 mV
Load resistance	100 Ω to 2 kΩ
x2, x5	+10 V ± 5 mV
Load resistance	300 Ω to 2 kΩ
EXCITATION LOAD REGULATION	
	15 ppm/mA
EXCITATION STABILITY	
	50 ppm/°C
EXCITATION PROTECTION	
	120 V ac
CMV (COMMON MODE VOLTAGE)	
Input to Output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	100 dB
NMR (NORMAL MODE REJECTION)	
	100 dB per decade above 8 kHz
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±25 ppm/°C
Gain	±100 ppm/°C
NOISE	
Output, 100 kHz	1500 µV rms
BANDWIDTH (-3 dB)	
	8 kHz
RESPONSE TIME	
90% span	70 µs
MODE 1	
Bandwidth	8 kHz
Input Range	-10 mV to +10 mV
Excitation	+3.333 V
Sensitivity	3 mV/V
MODE 2	
Bandwidth	8 kHz
Input Range	-30 mV to +30 mV
Excitation	+10.0 V
Sensitivity	3 mV/V

EX10SC-8B41-XX SPECIFICATIONS	
DESCRIPTIONS	
EX10SC-8B41-01	±1 V input with 1 kHz bandwidth
EX10SC-8B41-03	±10 V input with 1 kHz bandwidth
EX10SC-8B41-07	±20 V input with 1 kHz bandwidth
EX10SC-8B41-09	±40 V input with 1 kHz bandwidth
EX10SC-8B41-12	±60 V input with 1 kHz bandwidth
INPUT RANGE	
	±1 V to ±60 V
INPUT BIAS CURRENT	
	±0.5 nA
INPUT RESISTANCE	
Normal	500 kΩ minimum
Power off	500 kΩ minimum
Overload	500 kΩ minimum
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
CMV (COMMON MODE VOLTAGE)	
Input to Output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	100 dB
NMR (NORMAL MODE REJECTION)	
	100 dB per decade above 1 kHz
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±10 ppm/°C
Gain	±75 ppm/°C
NOISE	
Output, 100 kHz	500 µV rms
BANDWIDTH (-3 dB)	
	1 kHz
RESPONSE TIME	
90% span	550 µs
MTBF	
	596,000 hr

EX10SC-8B45-XX SPECIFICATIONS	
DESCRIPTIONS	
EX10SC-8B45-02	Frequency input (0 Hz to 1 kHz)
EX10SC-8B45-05	Frequency input (0 Hz to 10 kHz)
EX10SC-8B45-08	Frequency input (0 Hz to 100 kHz)
INPUT RANGE	
	0 Hz to 100 kHz
INPUT THRESHOLD (ZERO CROSSING)	
Minimum input	100 m V peak-to-peak
Maximum input	350 V peak-to-peak TTL
	170 V peak-to-peak zero-crossing
Minimum pulse width	4 µs
TTL input low	0.8V maximum
TTL input high	2.4V minimum
INPUT HYSTERESIS	
Zero crossing	±50 mV
TTL	1.5 V
INPUT RESISTANCE	
Normal	68 kΩ
Power off	68 kΩ
Overload	68 kΩ
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
EXCITATION	
	+5 V at 8 mA maximum
CMV (COMMON MODE VOLTAGE)	
Input to output	1500 V rms maximum
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	100 dB
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±25 ppm/°C
Gain	±100 ppm/°C
NOISE	
Output ripple	<10 m V peak-to-peak at input
	>2% span
BANDWIDTH (-3 dB)	
	1 kHz
RESPONSE TIME	
8B45-01	160 ms
8B45-02	80 ms
8B45-03	35 ms
8B45-04	16 ms
8B45-05	8.5 ms
8B45-06	3.4 ms
8B45-07	1.6 ms
8B45-08	0.8 ms

EX10SC-8B47X-XX SPECIFICATIONS	
DESCRIPTIONS	
EX10SC-8B47J-12	Linearized TC Type J (-100 °C to +760 °C)
EX10SC-8B47K-13	Linearized TC Type K (-100 °C to +1350 °C)
EX10SC-8B47T-06	Linearized TC Type T (-100 °C to +400 °C)
INPUT RANGE	
	-0.1 V to +0.5 V
INPUT BIAS CURRENT	
	-25 nA
INPUT RESISTANCE	
Normal	50 MΩ minimum
Power off	200 kΩ minimum
Overload	200 kΩ minimum
INPUT PROTECTION	
Continuous	240 V ac
Transient	ANSI/IEEE C37.90.1
CMV (COMMON MODE VOLTAGE)	
Input to Output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	120 dB
NMR (NORMAL MODE REJECTION)	
	70 dB at 60 Hz
STABILITY	
Offset	±20 ppm/°C
Gain	±75 ppm/°C
NOISE	
Output, 100 kHz	250 µV rms
BANDWIDTH (-3 dB)	
	3 Hz
RESPONSE TIME	
90% Span	150 ms
TRANSIENT	
	ANSI/IEEE C37.90.1
COLD JUNCTION COMPENSATION ACCURACY	
25 °C	±0.5 °C
–40 °C to +85 °C	±1.5 °C
OPEN INPUT RESPONSE	
	Upscale
OPEN INPUT DETECTION TIME	
	<10 s
MODEL 12	
TC type	J
Input range	-100 °C to +760 °C (–148 °F to +1400 °F)
Accuracy	±0.24% ± 2.10 °C
MODEL 13	
TC type	K
Input range	-100 °C to +1350 °C (–148 °F to +2462°F)
Accuracy	±0.24% ± 3.60 °C
MODEL 6	
TC type	T
Input range	-100 °C to +400 °C (–148 °F to +752°F)
Accuracy	±0.48% ± 2.40 °C
MTBF	
	520,000 hr

EX10SC-8B42-01 SPECIFICATIONS	
DESCRIPTION	
	2-wire transmitter interface
INPUT RANGE	
	4 mA to 20 mA
INPUT RESISTANCE	
Normal	35 Ω
Power off	35 Ω
INPUT PROTECTION	
Continuous	40 V ac
Transient	ANSI/IEEE C37.90.1
LOOP SUPPLY VOLTAGE	
	12 V dc
LOOP SUPPLY PROTECTION	
	40 V ac
CMV (COMMON MODE VOLTAGE)	
Input to Output	1500 V rms maximum
TRANSIENT	
Input to output	ANSI/IEEE C37.90.1
CMR (COMMON MODE REJECTION)	
50 Hz or 60Hz	100 dB
NMR (NORMAL MODE REJECTION)	
	60 dB per decade above 100Hz
ACCURACY	
	±0.05% span
LINEARITY	
	±0.02% span
STABILITY	
Offset	±25 ppm/°C
Gain	±100 ppm/°C
STABILITY	
Offset	±25 ppm/°C
Gain	±75 ppm/°C
NOISE	
Output, 100 kHz	500 µV rms
BANDWIDTH (-3 dB)	
	100 Hz
RESPONSE TIME	
90% span	5 ms

SECTION 10

EX10XXA/EX10XXA-51C/RX10XX/EX1044A/EX1044CALIBRATION

INTRODUCTION

It is recommended that the EX10xxA/EX10xxA-51C/RX10xx/EX1044A/EX1044 have factory calibration performed annually to ensure that the instrument maintains its accuracy and precision. The following procedure describes the resources and steps necessary to perform factory calibration and is valid for the following assemblies:

70-0355-100: EX1000A, 48-Ch Voltage Input Instrument
70-0355-200: EX1016A, 32-Ch Voltage, 16-Ch TC Input Instrument
70-0355-300: EX1032A, 16-Ch Voltage, 32-Ch TC Input Instrument
70-0355-400: EX1048A, 48-Ch TC Input Instrument
70-0355-500: EX1000A-TC, 48-Ch Voltage Input Instrument with TC Connectors
70-0564-000: EX1000A-51C, 51-Ch Voltage Input Instrument
70-0536-000: RX1032, 32-Ch Thermocouple/Voltage Instrument
70-0355-451: EX1044A, 48-Ch Thermocouple/Voltage/ Current (Isolated) Instrument
70-0355-452: EX1044, 44-Ch Thermocouple, w/4ch 5B module slot

REQUIRED RESOURCES

Equipment Needed

LAN/GPIB Gateway (Agilent E5810A recommended)
DMM (either an Agilent 3458A or Keithley 2002)
Test leads with banana jacks
One Ethernet switch
Three Ethernet cables (straight cables)
One GPIB cable

Software Needed

A web browser (Internet Explorer, Firefox, etc.)
The latest version of EX10xxA firmware, available on the VTI Instruments web site.
(The latest version of RX10xx firmware, available on the VTI Instruments web site).
Bonjour (optional: used to discover the EX10xxA/RX10xx, requires use of IE Version 6 or higher)
IO Library Suite (formerly called as Agilent Connection Expert) (optional: used for instrument discovery if Bonjour is not used. The latest version of IO Libraries can be downloaded from the Keysight website address i.e. http://www.keysight.com/find/iosuiteproductcounter)
Any NFS Server software (optional: used to collect the calibration log and data)

SETTING UP THE INSTRUMENTS

Prior to performing calibration, the EX10xxA/RX10xx must be connected to the required equipment as indicated in the steps below:

Fig 10-1, Demonstrates the EX10xxA unit for EX10xx calibration
Fig 10-2, Demonstrates the RX10xx unit for RX10xx calibration

1) Connect a GPIB cable from the LAN/GPIB gateway to the DMM.
2) Connect a banana jack probe from the Calibration Sense HI/LO jacks of EX10xxA/RX10xx to the HI/LO voltage terminals of DMM.
3) Connect an Ethernet cable from the PC's LAN port to the Ethernet switch.
4) Connect an Ethernet cable from the Ethernet switch to the LAN/GPIB gateway.
5) Power up all the instruments.
6) The GPIB/LAN Gateway should be able to query and list the attached GPIB devices through the browser. Note the GPIB address of connected DMM.

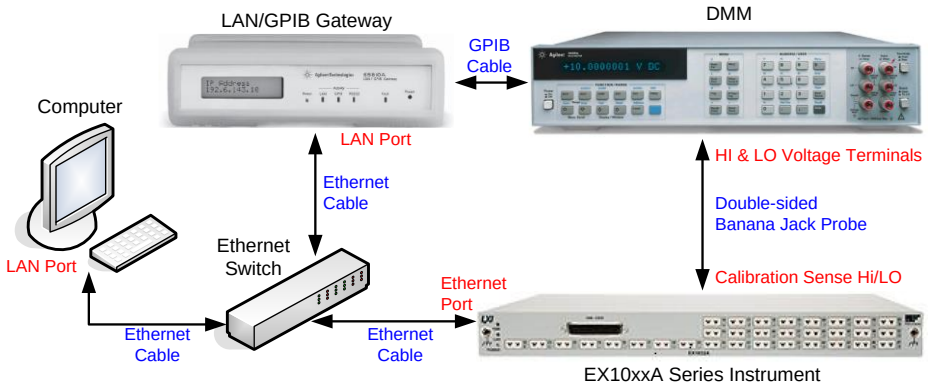


FIGURE 10-1: EX10XXA CABLING DIAGRAM

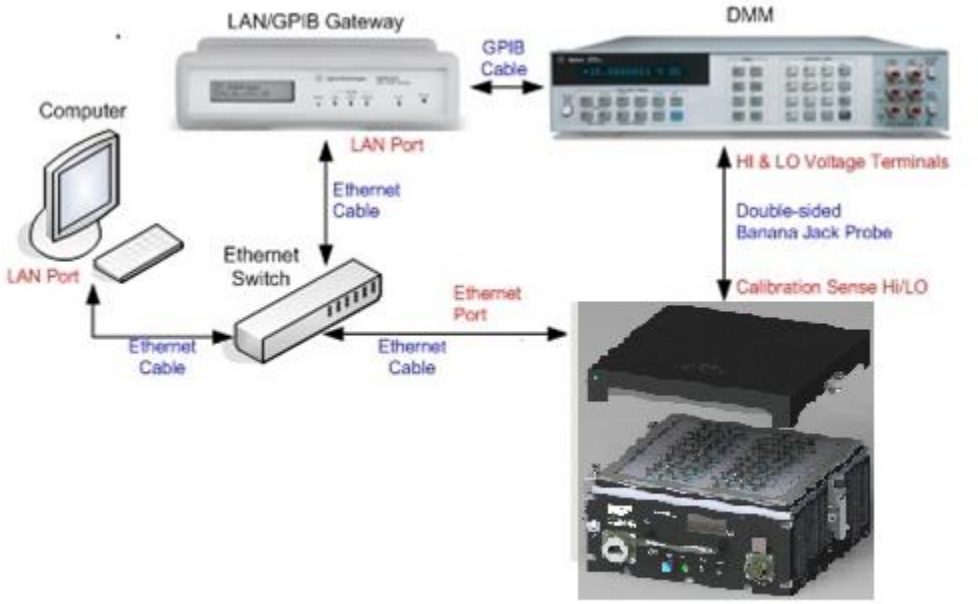


FIGURE 10-2: RX10XX CABLING DIAGRAM

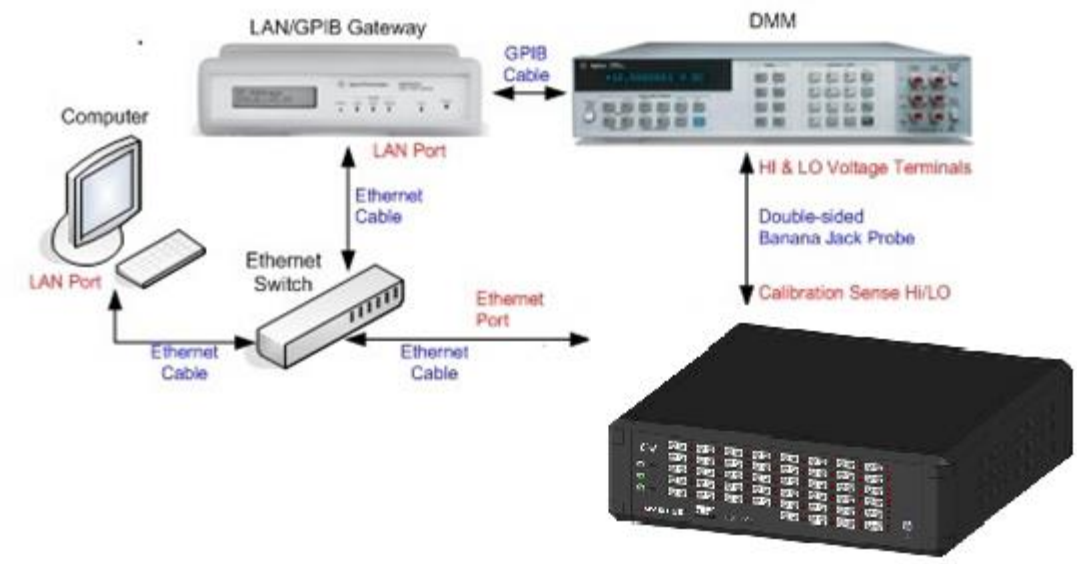


FIGURE 10-2: EX1044A/EX1044 CABLING DIAGRAM

NOTE	If the Agilent 3458A is used, the "Auto Cal All" procedure should be performed prior to calibrating the EX10xxA.

CONNECTING TO THE EX10XXA/RX10XX/EX1044A/EX1044

There are three ways to connect to the EX10xxA/RX10xx: using the IP address of the EX10xxA/RX10xx, using Bonjour for auto-discovery, or using the IO Library Suite (formerly as Agilent Connection Expert). For more information on obtaining the EX10xxA/RX10xx IP address and using this as the connection method, refer to the *Network Configuration* discussion in *Section 2*. To connect to the EX10xxA/RX10xx via Bonjour, refer to *Section 4: Web Page Operation*.

To use ACE, use the following steps:

1) Open ACE and click on the **Add instrument** toolbar.
2) Select **Add LAN instrument on LAN** and then click **OK**. ACE will search the network and list the LXI instruments connected to the LAN.

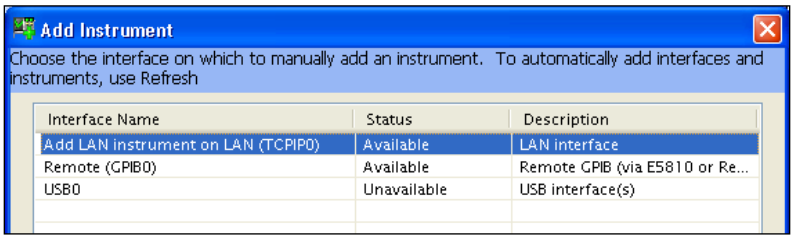


FIGURE 10-2: ADD INSTRUMENT INTERFACE

3) To simply view the EX10xxA/RX10xx web page, click on its webpage button. If desired, the EX10xxA/RX10xx can be added to ACE as a LAN instrument by enabling the appropriate **Select** checkbox and then clicking **OK**.

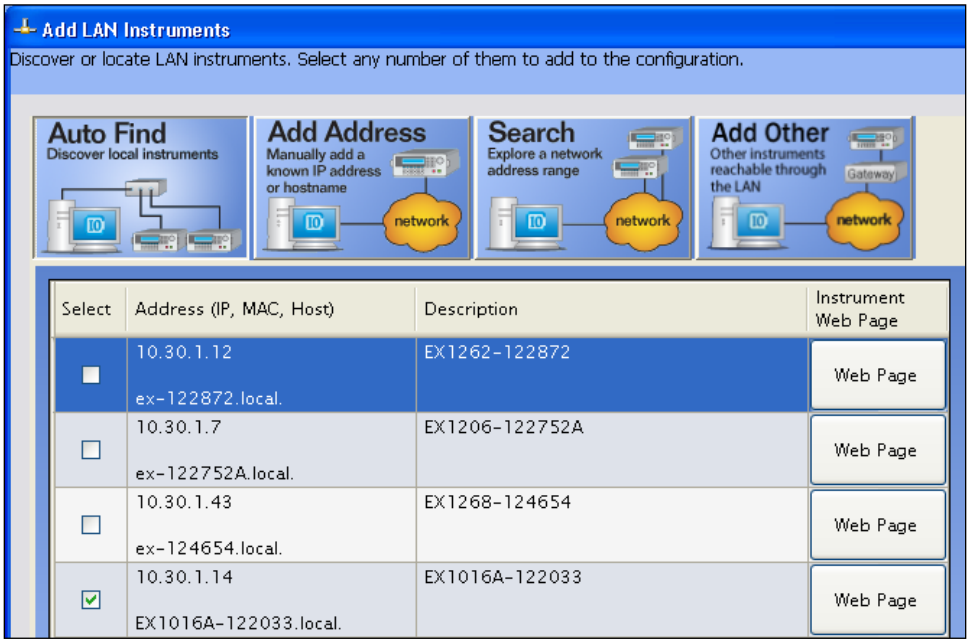


FIGURE 10-3: ACCESSING THE EX10XXA VIA ACE

LOGIN INTO SFP

EX10xxA/RX10xx calibration is performed using the SFP web interface. When accessing an action or entry page for the first time during a session, an **Error** page will appear. To clear this error, click on the **login** link. The **Login** page will appear. In the **Password** field, enter the configured instrument password. The default factory-programmed password is **ex1048**. For additional information, refer to the *General Web Page Operation* in *Section 4*.

NOTE	To reset the password to factory default, replace “index.cgi?” with “change_password.cgi?” in the browser’s address bar.

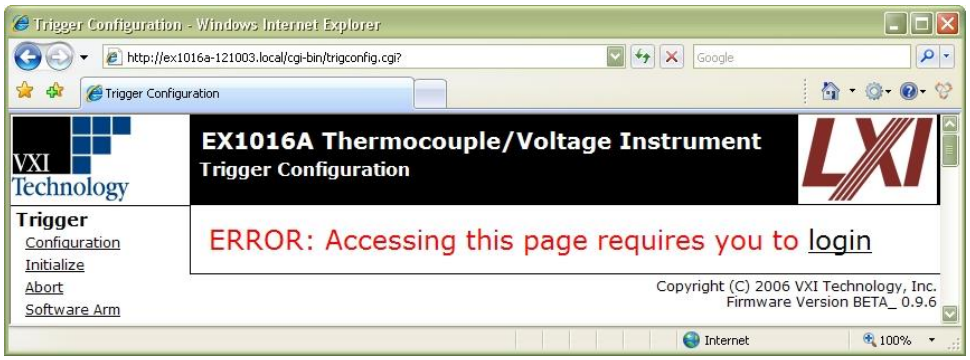


FIGURE 10-4: LOGIN ERROR PAGE

TIME CONFIGURATION

When calibration is performed, the time and date of calibration is recorded, making it important to ensure that the EX10xxA/RX10xx’s time is configured properly. To determine the time, click on the **Time Configuration** link in the command menu. If the time is displayed incorrectly, use the following steps to configure the time from the **Time Configuration** web page.

1) Select the correct **Time Zone** from the drop menu.

2) Set the **Time Source** to "Manual".
3) Enable the **Set Time** checkbox.
4) Enter the date in the **Date (MM:DD:YYYY)** fields.
5) Enter the time in time in the **Time (HH:MM:SS)** fields using a 24-hour format.
6) Click the **Submit** button.

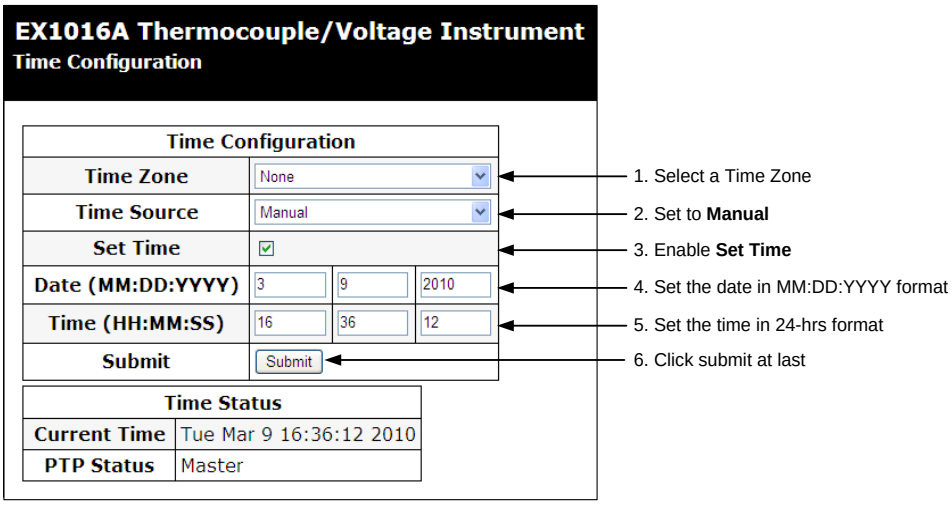


FIGURE 10-5: TIME CONFIGURATION

PERFORMING CALIBRATION

Once calibration equipment is connected to the EX10xxA/RX10xx and after ensuring that the time is set properly, it can be calibrated using the SFP. The following description provides how this is done.

1) In the browser's address bar, replace 'index.cgi?' with 'factory.cgi?' then press the **Enter** button. This will bring up the **Factory Mode Enable** page. Refer to the *Login into SFP* discussion for assistance in logging into the EX10xxA/RX10xx web page.

NOTE	1) A warning may appear indicating that a factory calibration file already exists. This warning can be ignored. 2) Calibration requires a one hour warm-up period. If the EX10xxA/RX10xx not has been on for one hour, a warning may appear and the calibration process will stop at this point.

2) When the **Enable Factory Mode** page loads, click the **Enable** button. The page will indicate that factory mode is enabled.

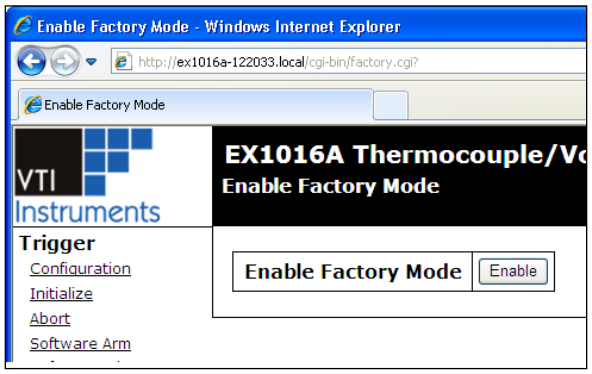


FIGURE 10-6: EX10XXA CALIBRATION PAGE

3) Once the Factory Mode is enabled successfully, in the browser's address bar, replace "factory.cgi" with "fullcal.cgi?" and then press **Enter** in order to access to the **Factory Calibration** page.
4) On the **Factory Calibration** page, select the appropriate DMM from the **DMM Model** drop menu.
5) Enter the LAN/GPIB Gateway's IP address in the **LAN/GPIB Gateway Address** field.
6) Enter the DMM's GPIB address (example: gpib0, 1) in the **GPIB Address** field. This can be obtained from LAN/GPIB Gateway's web page. Refer to the LAN/GPIB Gateway's user's manual for more information.
7) Click the **Submit** button to begin calibration.
8) Once calibration is complete, the calibration file can be retrieved from EX10xxA/RX10xx SFP using the **Get Calibration Files** link in the command menu.

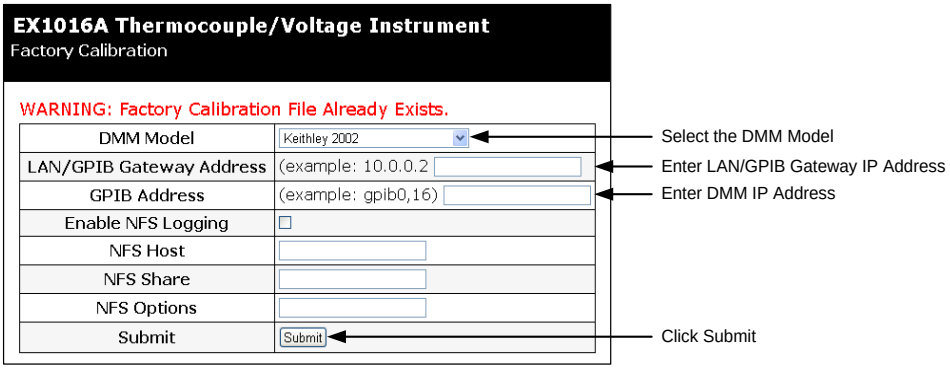


FIGURE 10-7: FACTORY CALIBRATION PAGE

NOTE	The NFS settings are optional. Enabling NFS will cause a log to be stored as well as a new calibration file on the NFS Host.

MONITORING CALIBRATION PROGRESS

Once the Submit button is clicked, the web page redirects to a status web page and will indicate that factory calibration has started successfully. If this page indicates an error during calibration, one of the previous steps may not have been followed correctly or there may be a problem with the EX10xxA/RX10xx. If a problem occurs during calibration, an error message is displayed on the screen next to the step that failed. If the progress reaches **Calibration Completed** and no steps report **Failed** or **Aborted**, then the factory calibration has completed successfully. If a **Failed**/**Aborted** status appears, the EX10xxA/RX10xx may require repair.

SECTION 11

ONBOARD MEMORY

ONBOARD MEMORY AND CLEARING PROCEDURE

The EX10xxA family of instruments contains onboard memory which stores various information about the unit as well as data acquired. Table 11-1 details the memory components and provided a procedure for clearing the memory.

Component	Volatile?	Contains	User Writeable?	Clear Procedure
64 Mb Flash (MFG: Micron Tech P/N: JS28F640J3F-75A)	No	Firmware	No	None
		File System	No	None
		Calibration Constants	Yes	Go to webpage, navigate to the **Self-calibration** page and click the **Clear Nonvolatile Self Cal Data** button Or Call the function: vtex10xxA_self_cal_clear_stored
		Network Configuration	Yes	Go to webpage, change values
		Time Configuration	Yes	Go to webpage, change values
8x16 MB SDRAM (MFG: Micron Tech P/N: MT48LC8M16A2P-7EG)	Yes	Runtime Data	Yes	Power cycle machine

TABLE 11-1: ONBOARD MEMORY AND CLEARING PROCESS

SECTION 12

QUICK REFERENCE GUIDE FOR RX1032

Mechanical Accessories

1) Power Connector: 27-0673-315, CONN, MIL-C-26482-1, CABLE PLUG, SIZE 14, 15 WAY, SOCKET
2) Ethernet Connector: 27-0926-008I, CONN,IP67,RJ-45,PL,8POS,ST
3) Mounting Angle Bracket: 41-0604-012 (2 ea)
4) Screw, M6X10mm: 37-1901-010 (6 ea)
5) Cable Mount Bracket: 41-0604-013 (2 ea)
6) Screw, M3X6mm: 37-0223-006 (4 ea)

NOTE:	Mechanical accessories need to order separately

Assembly Procedure

1) Mount option-I: 6X M6 tapped holes are provided at the bottom of the chassis as per Figure 12-1 for fixing on horizontal plate or with vibration mount.
2) Mount option-II: The System using Left and Right angle plates provided in the chassis.
3) Mount the unit to the protected area with mounting details provided as per Figure 92-1
4) Hole size in the mounting angle bracket is 6.5 mm (dia). Use 6 X M6 X 10mm LG, CAP FLANGE HD, 10.9 STEEL SCREWS

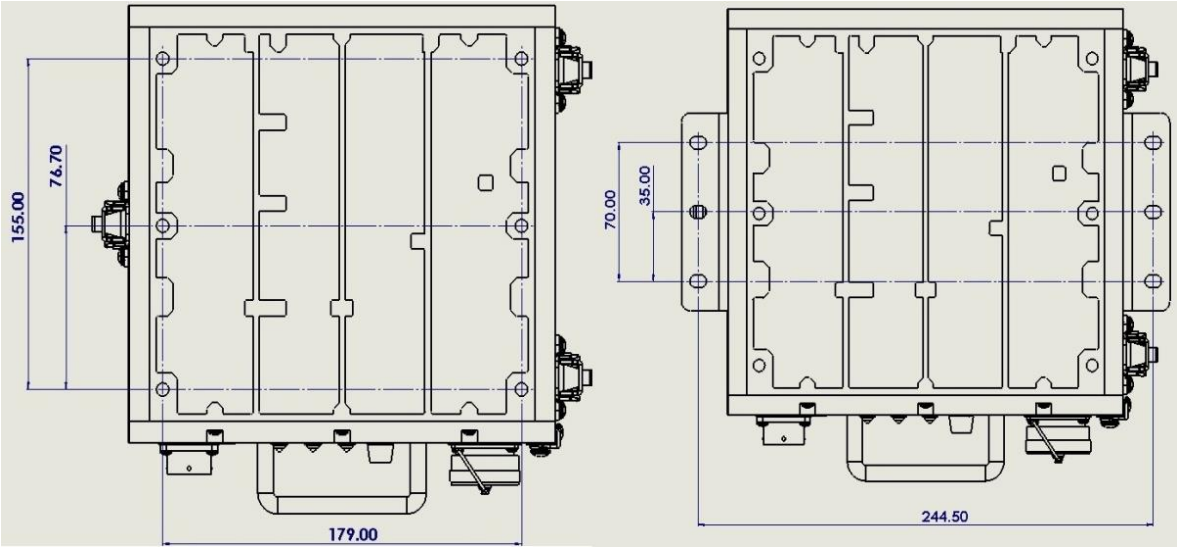


FIGURE 12-1: RX10XX CHASSIS MOUNTING DETAILS

5) Remove the Top Cover by opening 3 Latches on right and left side of chassis.



FIGURE 12-2: RX10XX CHANNEL SCREW TERMINAL PLATE VIEW

6) Unscrew the captive screws to inserts the thermocouple wires/ U - Lugs into it.
7) Fix the thermocouple 'U' Lugs into screw terminals. Each channel has S,+ & - signals screws (‘S’ for Shield, ‘+’ for Positive input, ‘-‘ for Negative input). Refer the below channel layout diagram of screw terminals.

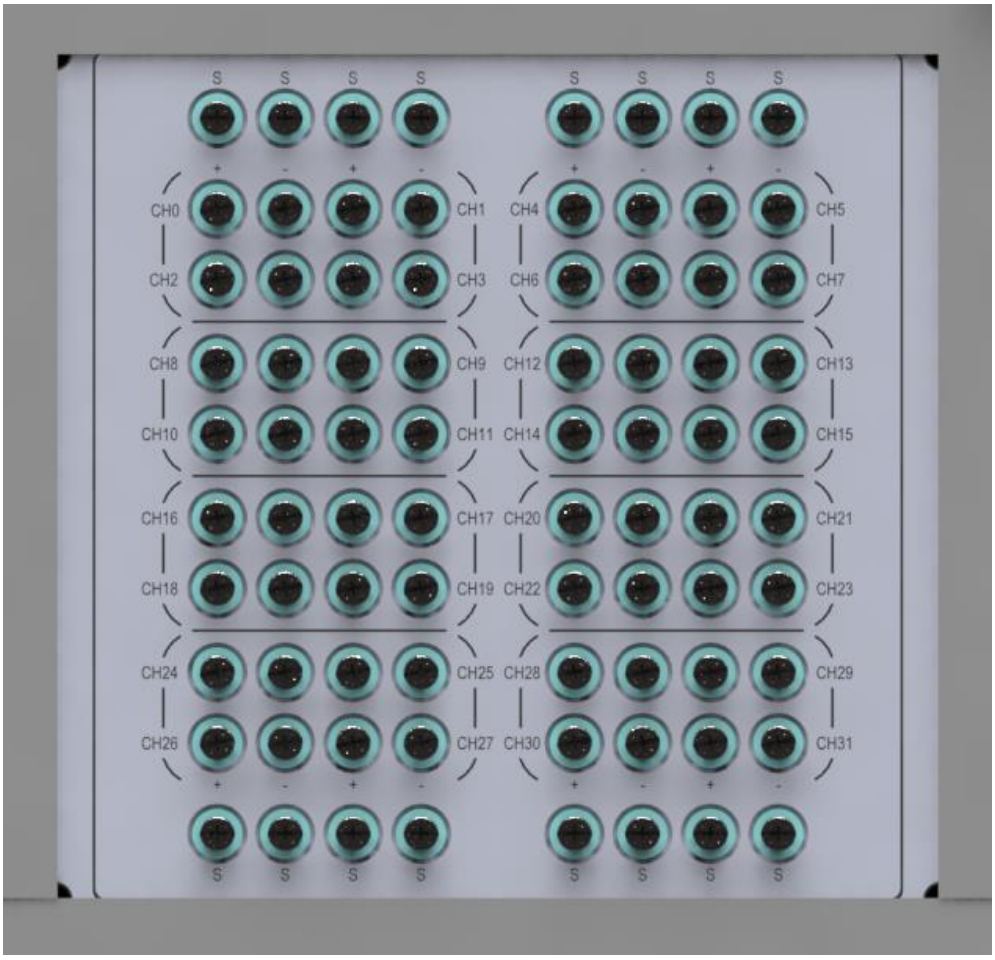
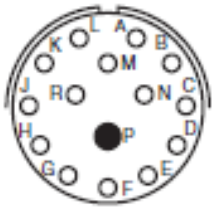


FIGURE 12-3: CHANNEL SCREW TERMINAL PLATE SHOWING 32 CHANNELS ON RX10XX



14-15

FIGURE 12-4: 15 PIN CIRCULAR CONNECTOR ON RX10XX (P1)

Pin	Description	Remarks
P1/R	+28V_IN	+28V DC Power Input
P1/P	28V_RET	28V Return DC Power Input
P1/P	28V_RET	28V Return DC Power Input
P1/N	+28V_IN	+28V DC Power Input
P1/A	DMM_HI_OUT	*Not to be Used (Factory Calibration Purpose)*
P1/L	DMM_LO_OUT	*Not to be Used (Factory Calibration Purpose)*

P1/B	**DAC0_OUT1**	***Not to be Used (Future Purpose)***
P1/K	**DAC0_OUT2**	***Not to be Used (Future Purpose)***
P1/C	**DAC1_OUT1**	***Not to be Used (Future Purpose)***
P1/J	**DAC1_OUT2**	***Not to be Used (Future Purpose)***
P1/M	**AGND**	***Not to be Used (Future Purpose)***
P1/D	**DIO1**	**General Purpose Digital I/O**
P1/H	**DIO2**	**General Purpose Digital I/O**
P1/E	**DIO3**	**General Purpose Digital I/O**
P1/G	**DIO4**	**General Purpose Digital I/O**
P1/F	**DGND**	**GND for General Purpose DIO**

TABLE 12-1: 13 PIN CIRCULAR CONNECTOR SIGNAL ASSIGNMENTS IN RX10XX (P1)

NOTE: 22 AWG wire can be used for the P1 connector cable looming



ATTENTION - Important safety instructions for RX10xx input supply

THE FOLLOWING PIN CONNECTION TO BE CAREFULLY CONNECTED WITH EXTERNAL SUPPLY. ANY MISTAKE IN THE CONNECTION MAY LEAD TO SYSTEM DAMAGE

Pin	Description
P1/R	**+28V_IN**
P1/P	**28V_RET**
P1/P	**28V_RET**
P1/N	**+28V_IN**

1) Connect P1 mating connector cable loom and other end +28V_IN and +28V_RET signals into 28V DC power supply. Power Supply should be capable of produce 2Amps
2) Power ON the supply.
3) Wait for 2min time to system boot.
4) Run the SFP application software and configure the channel as per requirement.
5) Wait for warm up time as mentioned in "WARMUP TIME" section
6) Start acquire the channel data

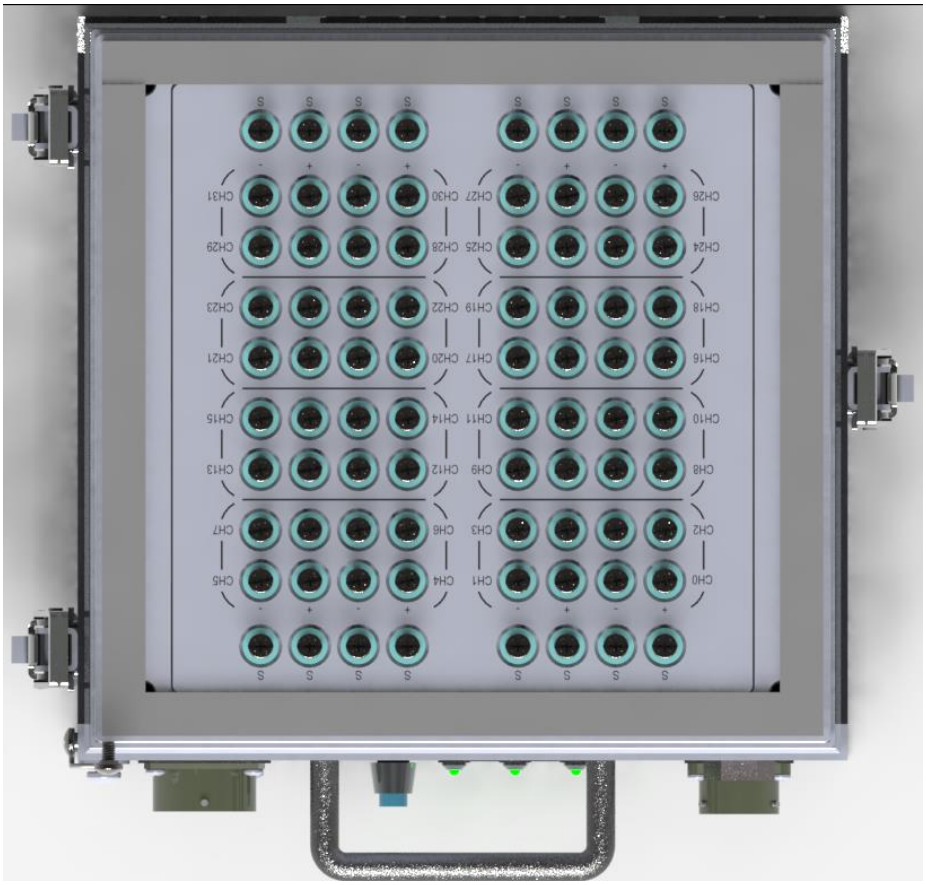


FIGURE 12-5: RX1032 CHANNEL SCREW TERMINAL PLATE VIEW



FIGURE 12-6: RX1032 FULL VIEW

SECTION 13

QUICK REFERENCE GUIDE FOR EX1044A/EX1044

Assembly

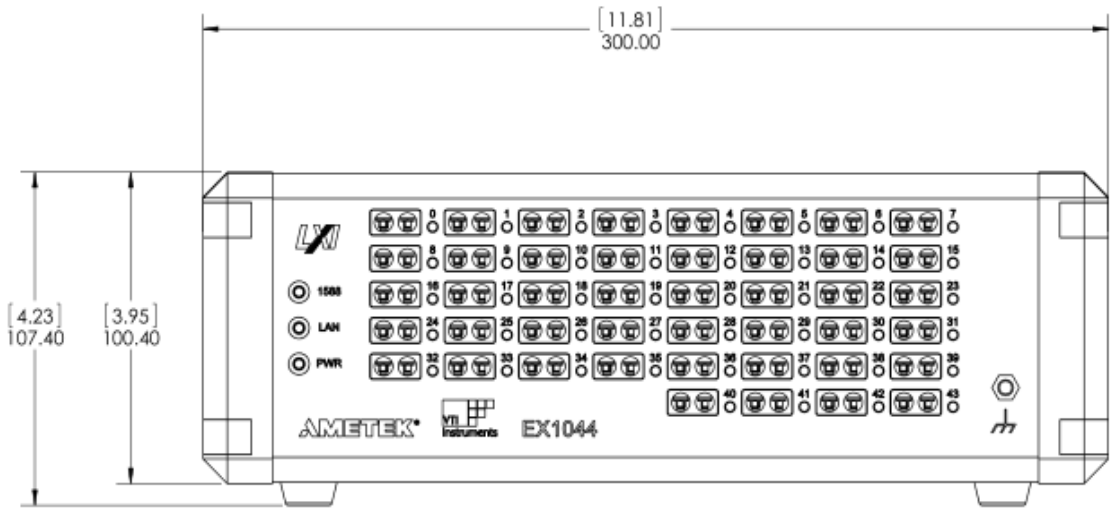


FIGURE 13-1: TABLE TOP USAGE

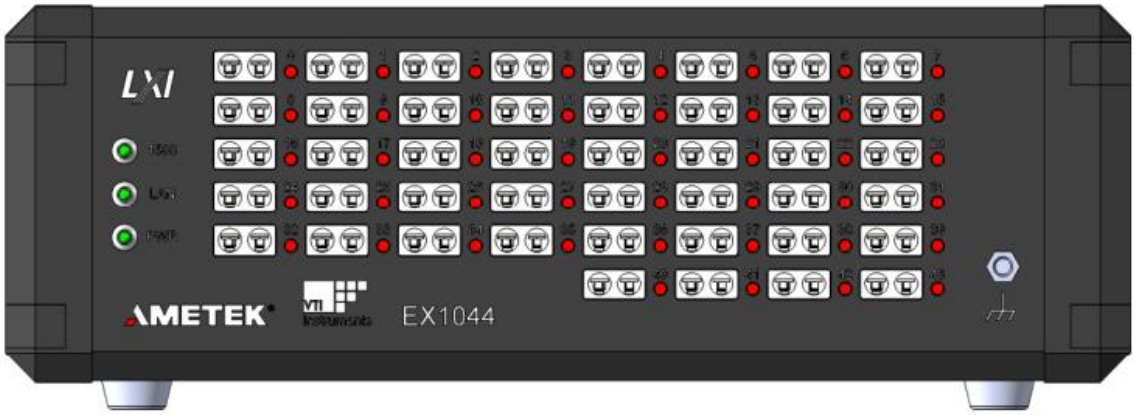


FIGURE 13-2: EX1044A/EX1044 TABLE TOP MOUNT (FRONT VIEW)



FIGURE 13-3: EX1044A/EX1044 TABLE TOP MOUNT (BACK VIEW)



FIGURE 13-2: EX1044A/EX1044 (ISOMETRIC VIEW)

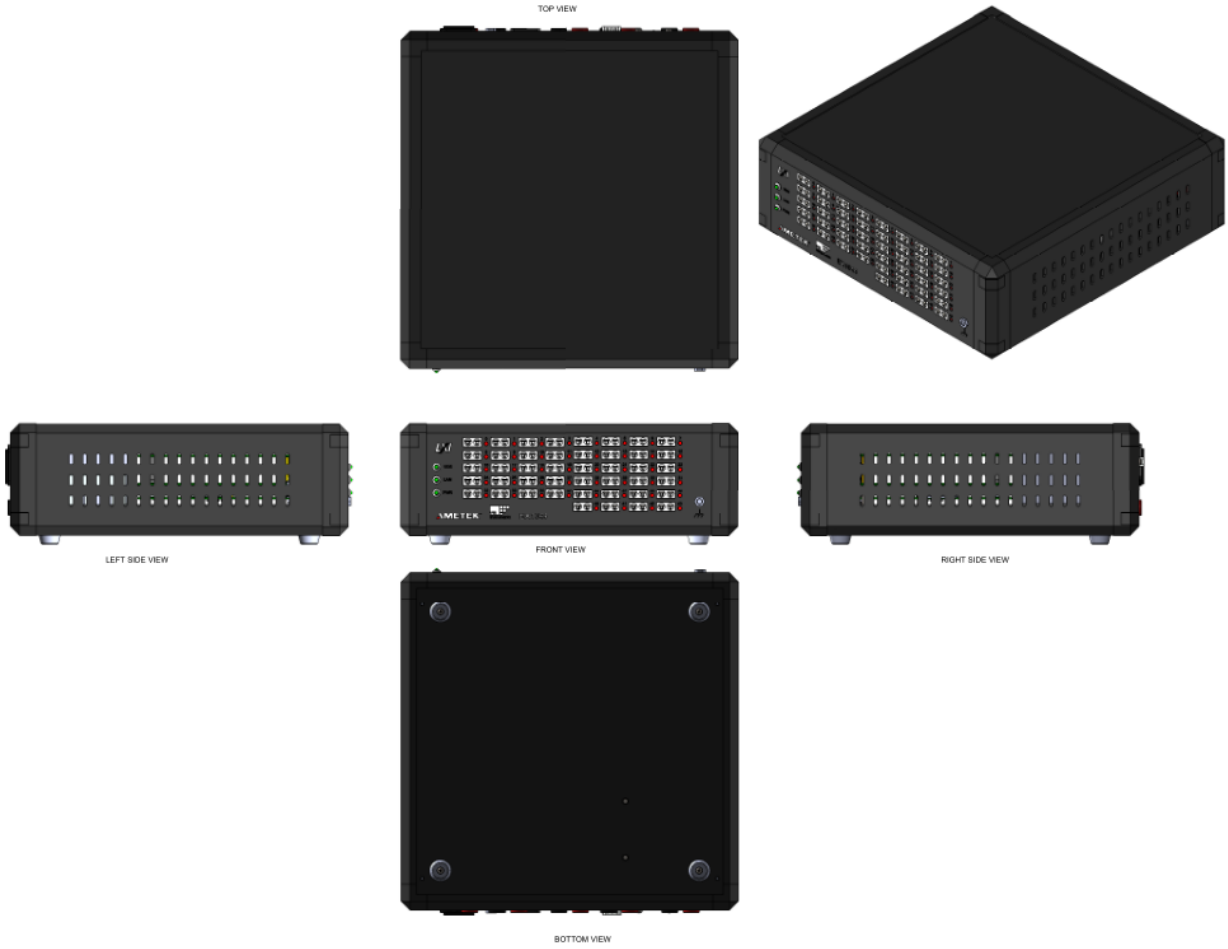


FIGURE 13-3: EX1044A/EX1044 (DIFFERENT VIEW)

INDEX

A

acquiring data 67, 79, 113
advanced menu 94
alarm ..... *See* LXI alarm
ARM event ..... *See* triggering
AutoIP 45, 89

B

blocking mode 68, 84, 105
Bonjour 47, 48

C

calibration ..... *See* self-calibration
CJC ..... See cold junction compensation
CMRR 20, 58
cold junction compensation 17, 31, 59, 83, 94, 103, 283
common mode input range 20, 58
common mode rejection ratio ..... *See* CMRR
connectors
  trigger bus 65

D

data
  format 67, 83, 105
  menu 83
declaration of conformity 10, 11
default settings 98
device identify 90
device menu 87
DHCP server 45, 89
digital I/O 18, 63, 65, 67, 71, 72, 84, 85
dimensions 20
DIO limit events 63, 84, 85, 108

E

engineering unit (EU) conversion 57, 102
error messages 278
EX10SC 285
  overview 285
EX10SC module specifications ..295–310, 295–310, 295–310, 295–310
  EX10SC-8B32-02 295
  EX10SC-8B33-03 298
  EX10SC-8B33-04 298
  EX10SC-8B33-05 298
  EX10SC-8B34-04 296
  EX10SC-8B35-04 299
  EX10SC-8B36-04 297
  EX10SC-8B38-01 300
  EX10SC-8B38-02 300
  EX10SC-8B41-01 301
  EX10SC-8B41-03 301
  EX10SC-8B41-07 301
  EX10SC-8B41-09 301
  EX10SC-8B41-12 301
  EX10SC-8B42-01 304
  EX10SC-8B45-02 302
  EX10SC-8B45-05 302
  EX10SC-8B45-08 302
  EX10SC-8B475-13 303
  EX10SC-8B47J-12 303
  EX10SC-8B47T-06 303

F

features 16
FIFO
  acquiring data into 67
  clearing 80, 93, 100, 113
  configuration 83, 105
  get count of 83, 115
  retrieving data from 68, 83, 115
filter 18, 58, 83, 104
firmware
  upgrade 93
function return value 127
function set 127

I

input connector 18, 20, 31, 51
input protection 20, 58
IO menu 84

L

LAN event 71, 80, 109
  configuration 80
  log 81
LEDs 63
limits 63, 85, 105
limits menu 85
locking 67, 94, 101
locking menu 94
LXI alarm 67, 71, 72, 147, 148, 223, 224
LXI Trigger Bus 65

M

MAC address 45, 89
measurement range 17, 58, 59
memory ..... *See* FIFO
memory clearing 311

N

network configuration 45, 89
  troubleshooting 46
NIST ITS-90 thermocouple database 30, 57, 69, 283
noise performance 18, 33, 52, 58, 60, 72
NTP 91

O

onboard memory 311, 312, 317
open transducer detection 16, 63, 282
OTD detection ..... See open transducer detection

R

ranges 83
Required Resources 305
reset
  button 45, 89
  device 93, 100

network configuration 45, 89
retreiving data....115
asynchronous streaming data interface 118
retrieving data....68, 83

S

sampling rate 18, 19, 33, 60, 61, 72
scan list
configuration 61, 82
menu....82
timing 61
self-calibration.... 30, 31, 62, 87, 88, 100, 283
SNTP....91
software installation....43
specifications....19, 20, 21, 30, 62
environmental....21
streaming data
basic 120
callback function 121

T

temperature accuracy.... *See* specifications
temperature units....60, 83, 105
thermocouple calculations....57, 69, 283
thermocouple wire....*See* wiring
time configuration 51, 90
timestamps....67, 83, 105
TRIG event....*See* triggering
trigger bus (VTB) 19, 65, 66, 67, 71, 72, 84, 109, 111
trigger menu 79
triggering....19, 67, 70, 79, 109
troubleshooting
multiple network cards 46, 49

U

units....*See* temperature units
user-defined conversions 57, 69, 94, 103
using multiple network cards....49

V

voltage....19
vtex10xxA_abort....133
vtex10xxA_append_scanlist....134
vtex10xxA_break_lock....135
vtex10xxA_check_lock 136
vtex10xxA_clear_lan_eventlog 137
vtex10xxA_close....138
vtex10xxA_disable_streaming_data....139
vtex10xxA_enable_streaming_data....140
vtex10xxA_enable_streaming_dataEx....142
vtex10xxA_error_message 144
vtex10xxA_error_query....145
vtex10xxA_get_accum_limit_status....146
vtex10xxA_get_alarm 147
vtex10xxA_get_alarm_enable 148
vtex10xxA_get_arm_count....149
vtex10xxA_get_arm_delay....150
vtex10xxA_get_arm_infinite....151
vtex10xxA_get_arm_lan_eventID....152
vtex10xxA_get_arm_lan_filter....153
vtex10xxA_get_arm_source....154
vtex10xxA_get_arm_sourceEx....155
vtex10xxA_get_calibration_file 156
vtex10xxA_get_calibration_running 157
vtex10xxA_get_channel_conversion....158, 159
vtex10xxA_get_channel_range 160
vtex10xxA_get_channel_type 161
vtex10xxA_get_dio_input 162
vtex10xxA_get_dio_limit_event 163
vtex10xxA_get_dio_limit_event_invert....164
vtex10xxA_get_dio_limit_event_latch 165
vtex10xxA_get_dio_output....166
vtex10xxA_get_dio_output_enable....167
vtex10xxA_get_fifo_config 168
vtex10xxA_get_fifo_count 169
vtex10xxA_get_filt_freq....170
vtex10xxA_get_init_cont....171
vtex10xxA_get_lan_event_domain....172
vtex10xxA_get_lan_event_source_state 173
vtex10xxA_get_lan_eventlog_count....174
vtex10xxA_get_lan_eventlog_enabled 175
vtex10xxA_get_lan_eventlog_overflowmode....176
vtex10xxA_get_limit_set0 177
vtex10xxA_get_limit_set0_manual....178
vtex10xxA_get_limit_set1 179
vtex10xxA_get_linear_correction....180
vtex10xxA_get_model 181
vtex10xxA_get_OTD_enable....182
vtex10xxA_get_ptp_info....183
vtex10xxA_get_scanlist....184
vtex10xxA_get_serialNumber 128, 185
vtex10xxA_get_system_time....187
vtex10xxA_get_trig_lan_eventID....188
vtex10xxA_get_trig_lan_filter....189
vtex10xxA_get_trigger_count....190
vtex10xxA_get_trigger_delay....191
vtex10xxA_get_trigger_infinite....192
vtex10xxA_get_trigger_source 193
vtex10xxA_get_trigger_sourceEx....194
vtex10xxA_get_trigger_timer 195
vtex10xxA_get_user_cjc_enable....196
vtex10xxA_get_user_cjc_temp....197
vtex10xxA_get_user_conversion 198
vtex10xxA_get_vtb_input....199
vtex10xxA_get_vtb_output....200
vtex10xxA_get_vtb_output_enable....201
vtex10xxA_get_vtb_wiredor....202
vtex10xxA_init 204
vtex10xxA_init_imm 205
vtex10xxA_lock....206
vtex10xxA_pop_logged_LAN_event....207
vtex10xxA_read_fifo 208
vtex10xxA_read_fifoEx....209
vtex10xxA_reset 210
vtex10xxA_reset_fifo....211
vtex10xxA_reset_trigger_arm....212
vtex10xxA_revisionQuery 213
vtex10xxA_self_cal_clear....214
vtex10xxA_self_cal_clear_stored 215
vtex10xxA_self_cal_get_status....216
vtex10xxA_self_cal_init 217
vtex10xxA_self_cal_is_stored 218
vtex10xxA_self_cal_load....219
vtex10xxA_self_cal_store....220
vtex10xxA_self_test 129, 221, 222, 225
vtex10xxA_set_alarm 223
vtex10xxA_set_alarm_enable 224
vtex10xxA_set_arm_count 226
vtex10xxA_set_arm_delay....227
vtex10xxA_set_arm_infinite....228
vtex10xxA_set_arm_lan_eventID....229
vtex10xxA_set_arm_lan_filter....230
vtex10xxA_set_arm_source....231
vtex10xxA_set_arm_sourceEx....232
vtex10xxA_set_channel_conversion....233
vtex10xxA_set_channel_range 234
vtex10xxA_set_dio_limit_event 236
vtex10xxA_set_dio_limit_event_invert 237
vtex10xxA_set_dio_limit_event_latch....238
vtex10xxA_set_dio_output 239
vtex10xxA_set_dio_output_enable 240

vtex10xxA_set_dio_pulse....241
vtex10xxA_set_fifo_config....242
vtex10xxA_set_filt_freq....243
vtex10xxA_set_init_cont....244
vtex10xxA_set_lan_event_domain....245
vtex10xxA_set_lan_eventlog_enabled....246
vtex10xxA_set_lan_eventlog_overflowmode....247
vtex10xxA_set_limit_set0....248
vtex10xxA_set_limit_set0_manual....249
vtex10xxA_set_limit_set1....250
vtex10xxA_set_linear_correction....251
vtex10xxA_set_OTD_enable....252
vtex10xxA_set_scanlist....253
vtex10xxA_set_trig_lan_eventID....254
vtex10xxA_set_trig_lan_filter....255
vtex10xxA_set_trig_source_timer....256
vtex10xxA_set_trigger_count....257
vtex10xxA_set_trigger_delay....258
vtex10xxA_set_trigger_infinite....259
vtex10xxA_set_trigger_source....260
vtex10xxA_set_trigger_sourceEx....261
vtex10xxA_set_trigger_timer....262
vtex10xxA_set_user_cjc_enable....263
vtex10xxA_set_user_cjc_temp....264
vtex10xxA_set_user_conversion....265
vtex10xxA_set_vtb_output....266
vtex10xxA_set_vtb_output_enable....267
vtex10xxA_set_vtb_pulse....268
vtex10xxA_set_vtb_wiredor130, 186, 235, 269, 274, 275, 276, 277
vtex10xxA_set_vtb_wiredor_bias....270
vtex10xxA_soft_arm....271
vtex10xxA_soft_trigger....272
vtex10xxA_unlock....273
VXI-11 device discovery....45, 89

W

warm-up....22, 43, 62, 87, 100
web page
  LAN event....80
WEEE....12
wiring....51

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP