

# ***GX3348***

## **Multifunction DC Source and Switch Matrix PXI Board GXPDO Software**

### ***User's Guide***

Last updated: March 21, 2016



## Safety and Handling

---

Each product shipped by Marvin Test Solutions is carefully inspected and tested prior to shipping. The shipping box provides protection during shipment, and can be used for storage of both the hardware and the software when they are not in use.

The circuit boards are extremely delicate and require care in handling and installation. Do not remove the boards from their protective plastic coverings or from the shipping box until you are ready to install the boards into your computer.

If a board is removed from the computer for any reason, be sure to store it in its original shipping box. Do not store boards on top of workbenches or other areas where they might be susceptible to damage or exposure to strong electromagnetic or electrostatic fields. Store circuit boards in protective anti-electrostatic wrapping and away from electromagnetic fields.

Be sure to make a single copy of the software CD for installation. Store the original CD in a safe place away from electromagnetic or electrostatic fields. Return compact disks (CD) to their protective case or sleeve and store in the original shipping box or other suitable location.

## Warranty

---

Marvin Test Solutions products are warranted against defects in materials and workmanship for a period of 12 months. Marvin Test Solutions shall repair or replace (at its discretion) any defective product during the stated warranty period. The software warranty includes any revisions or new versions released during the warranty period. Revisions and new versions may be covered by a software support agreement. If you need to return a board, please contact Marvin Test Solutions Customer Technical Services Department via <http://www.marvintest.com/magic/> - the Marvin Test Solutions on-line support system.

## If You Need Help

---

Visit our web site at <http://www.marvintest.com> for more information about Marvin Test Solutions products, services and support options. Our web site contains sections describing support options and application notes, as well as a download area for downloading patches, example, patches and new or revised instrument drivers. To submit a support issue including suggestion, bug report or questions please use the following link: <http://www.marvintest.com/magic/>

You can also use Marvin Test Solutions technical support phone line (949) 263-2222. This service is available between 8:30 AM and 5:30 PM Pacific Standard Time.

## Disclaimer

---

In no event shall Marvin Test Solutions or any of its representatives be liable for any consequential damages whatsoever (including unlimited damages for loss of business profits, business interruption, loss of business information, or any other losses) arising out of the use of or inability to use this product, even if Marvin Test Solutions has been advised of the possibility for such damages.

## Copyright

---

Copyright © 2012-2016 by Marvin Test Solutions., All rights reserved. No part of this document can be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Marvin Test Solutions.

## Trademarks

---

ATEasy®, CalEasy, DIOEasy®, DtifEasy, WaveEasy	Marvin Test Solutions (prior name is Geotest - Marvin Test Systems Inc.)
C++ Builder, Delphi	Embarcadero Technologies Inc.
LabView, LabWindows™/CVI	National Instruments
Microsoft Developer Studio, Microsoft Visual C++, Microsoft Visual Basic, .NET, Windows 95, 98, NT, ME, 2000, XP, VISTA, Windows 7, 8 and 10	Microsoft Corporation

All other trademarks are the property of their respective owners.

## Table of Contents

Safety and Handling.....	i
Warranty .....	i
If You Need Help.....	i
Disclaimer .....	i
Copyright .....	i
Trademarks .....	ii
<b>Table of Contents.....</b>	<b>iii</b>
<b>Chapter 1 - Introduction .....</b>	<b>1</b>
Manual Scope and Organization.....	1
Manual Scope.....	1
Manual Organization.....	1
Conventions Used in this Manual .....	1
<b>Chapter 2 - Overview .....</b>	<b>3</b>
Introduction.....	3
Features.....	3
Applications.....	3
Board Description .....	4
Architecture .....	6
Output Channels.....	7
Specifications.....	8
Virtual Panel Description.....	9
Virtual Panel Initialize Dialog .....	10
Virtual Panel Setup Page.....	11
Virtual Panel About Page.....	13
<b>Chapter 3 - Installation and Connections .....</b>	<b>15</b>
Getting Started .....	15
Packing List .....	15
Unpacking and Inspection.....	15
System Requirements.....	15
Installation of the GXPDO Software .....	16
Overview of the GXPDO Software .....	17
Configuring Your PXI System using the PXI/PCI Explorer.....	18
Board Installation.....	19
Before you Begin .....	19
Electric Static Discharge (ESD) Precautions .....	19

Installing a Board .....	19
Plug & Play Driver Installation .....	20
Removing a Board .....	21
Connectors and Accessories .....	21
Connectors .....	22
Installation Folders .....	23
GXPDO Driver Files Description .....	23
Driver File and Virtual Panel .....	23
Interface Files.....	23
On-line Help and Manual .....	24
ReadMe File.....	24
Example Programs .....	24
Setup Maintenance Program .....	25
<b>Chapter 4 - Programming the Board .....</b>	<b>27</b>
The GXPDO Driver .....	27
Programming Using C/C++ Tools .....	27
Programming Using Visual Basic.....	27
Programming Using C# .....	27
Programming Using Pascal/Delphi .....	28
Programming Using ATEasy® .....	28
Programming Using LabView and LabView/Real Time .....	28
Using and Programming under Linux.....	28
Using the GXPDO Driver Functions .....	29
Initialization, HW Slot Numbers and VISA Resource .....	29
Board Handle .....	30
Reset.....	30
Error Handling .....	30
Driver Version.....	30
Panel.....	30
Distributing the Driver .....	31
Sample Programs .....	31
Sample Program Listing .....	32
<b>Chapter 5 - Calibration.....</b>	<b>39</b>
Introduction.....	39
Hardware Requirements.....	39
Calibration Licensing.....	39
GXPDO Functions used for Calibration .....	41

Calibration Procedure .....	42
Initial Calibration Procedure .....	42
DAC Voltage Calibration.....	42
ADC Calibration .....	42
Final Calibration Procedure .....	42
<b>Chapter 6 - Functions Reference.....</b>	<b>43</b>
Introduction.....	43
GX3348 Functions .....	44
Gx3348CalAdc .....	45
Gx3348CalDac.....	47
Gx3348CalSetDacVoltagePoint .....	48
Gx3348CalSetMode.....	50
Gx3348CalWriteEEPROM.....	51
Gx3348GetBoardSummary.....	52
Gx3348GetCalibrationInfo .....	53
Gx3348GetChannelRail .....	55
Gx3348GetDac .....	56
Gx3348GetRailSource .....	57
Gx3348GetDigitalInputs.....	59
Gx3348GetDigitalOutputs .....	60
Gx3348Initialize .....	61
Gx3348InitializeVisa .....	62
Gx3348Measure.....	63
Gx3348Panel.....	64
Gx3348Reset.....	65
Gx3348SelfTest .....	66
Gx3348SetChannelRail .....	67
Gx3348SetDac .....	68
Gx3348SetRailSource.....	69
Gx3348SetDigitalOutputs.....	72
GxPdoGetDriverSummary.....	73
GxPdoGetErrorString .....	74
<b>Index .....</b>	<b>77</b>





# Chapter 1 - Introduction

## Manual Scope and Organization

### Manual Scope

This manual provides all the information necessary for installation, operation, and maintenance of the GX3348 Multi-Channel Analog I/O card. This manual assumes the reader has a general knowledge of PC based computers, Windows operating systems, and familiarity with analog instrumentation.





This manual also provides programming information about using the GX3348 driver (referred in this manual **GXPDO**). This manual assumes the user has a thorough understanding of Windows application development tools and languages.

### Manual Organization

The GX3348 manual is organized in the following manner:

Chapter	Content
Chapter 1 – Introduction	Introduces the GX3348 manual. Lists all the supported boards and shows warning conventions used in the manual.
Chapter 2 – Overview	Provides the GX3348 list of features, description of the board, architecture, specifications and the virtual panel description and operation.
Chapter 3 – Installation and Connections	Provides instructions on how to install a GX3348 board and the GXPDO software.
Chapter 4 – Programming the Board	Provides a list of the GXPDO software driver files, general purpose and generic driver functions, and programming methods. Discusses supported application development tools and programming examples.
Chapter 5 – Calibration	Describes the procedure used to perform calibration.
Chapter 6 – Functions Reference	Provides a list of the GXPDO driver functions. Each function description provides syntax, parameters, and any special programming comments.

## Conventions Used in this Manual

Symbol Convention	Meaning
	Static Sensitive Electronic Devices. Handle Carefully.
	Warnings that may pose a personal danger to your health. For example, shock hazard.
	Cautions where computer components may be damaged if not handled carefully.
	Tips that aid you in your work.

<b>Formatting Convention</b>	<b>Meaning</b>
Monospaced Text	Examples of field syntax and programming samples.
<b>Bold type</b>	Words or characters you type as the manual instructs. For example: function or panel names.
<i>Italic type</i>	Specialized terms. Titles of other references and information sources. Placeholders for items you must supply, such as function parameters

## Chapter 2 - Overview

### Introduction

---

The GX3348 is a 6U PXI Analog input and output card and has Precision DC Source to support the output function. The GX3348 supports two configurations, GX3348-1 with 48 channels and the GX3348-2 with 64 channels. The GX3348 has three analog rails and additional ground rail that can be connected to a full matrix, allowing any of the four rails to be routed to any of the 48 outputs (GX3348-1) or 64 outputs (GX3348-2). Each of the three analog rails has a dedicated programmable high current DC voltage source that can be programmed from -20V to +32V with 1mV resolution. The source of the rails can be set programmatically to a dedicated programmable high current DC voltage source, external input, the chassis' 5V power supply, or amplified external input.

Each of the 48 or 64 channels includes DC measurement capability and can be used as an input. Channel measurements can be performed using an on-board 16 bit A to D converter. In addition, there are eight general-purpose digital I/O lines.

### Features

---

- 48/64 (model dependent) analog channels which can be connected to any of four rails. Each channel can be measured using an on-board A to D converter at any time.
- Three multifunction, 16-bit DC sources, with a -20V to +32V range and maximum current of 500mA.
- Rail A accesses a precision DC source, external reference, or an on-board amplifier's output which has a nominal gain of 6.6.
- Rail B accesses a precision DC source or a +5 V power supply.
- Rail C accesses a precision DC source or an external reference source.
- Rail D is connected to ground (0 V)
- Programmable 8 independent digital I/O lines.
- Access to all signals is via a 78-pin D-Type connector.
- A software package GXPDO provides programming and examples for interfacing to common programming languages under 32/64 bit Windows. A front panel application is also provided which displays and controls the instrument's functions. A separate software package (**GtLinux**) provides support for Linux.
- The calibration data source for each DAC is set by software and is stored in non-volatile memory on the card. Calibration for each DAC can be done using a software API or using **CalEasy** – Marvin Test Solutions' Calibration executive (both the API and CalEasy require the purchase of a software license).

### Applications

---

- Simulation of discrete levels, common in avionics testing.
- UUT power
- Sensor emulation and control.

## Board Description

---

The GX3348 supports two configurations, GX3348-1 with 48 channels and the GX3348-2 with 64 channels. The GX3348 has three analog rails and an additional ground rail that can be connected the full matrix, allowing any of the four rails to be routed to any of the 48 outputs (GX3348-1) or 64 outputs (GX3348-2). Each of the three analog rails has a dedicated programmable high current DC voltage source that can be programmed from -20V to +32V with 1mV resolution. The source of the rails can be set programmatically to a dedicated programmable high current DC voltage source, an external input, the chassis' 5V power supply, or to the on-board amplifier's output.

The GX3348 has four rails (Rail A, Rail B, Rail C and Ground Rail) that can be connected to various sources. Each rail has a set of resources that it can be accessed as follows.

Sources for Rail A:

- High-resolution, programmable high current voltage source. Voltage range is -20V to +32V @ 500mA.
- External source via the 78 pin front panel connector.
- External input via the on-board amplifier, which has a nominal gain of 6.6.

Sources for Rail B:

- High-resolution, programmable high current voltage source. Voltage range is -20V to +32V @ 500mA.
- +5 volt power supply

Sources for Rail C:

- High-resolution, programmable high current voltage source. Voltage range is -20V to +32V @ 500mA.
- External source via the 78-front panel connector.

Source for Ground Rail:

- Ground (0V)

Figure 2-1 shows the GX3348 with its J3 connector.

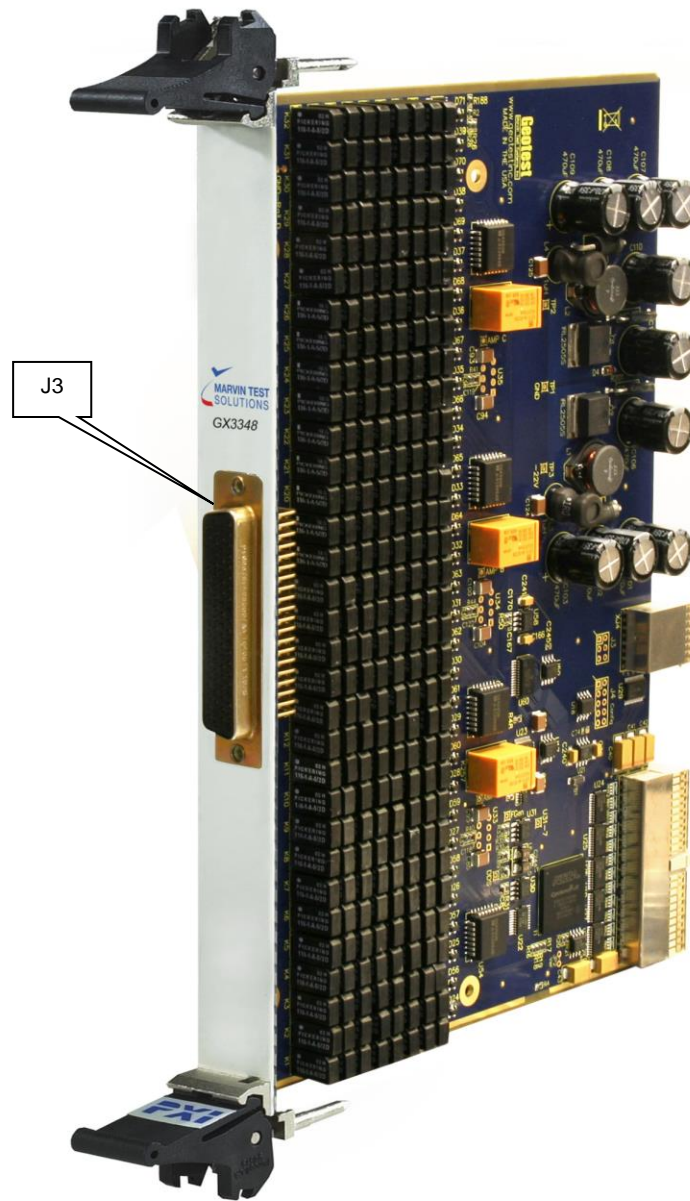
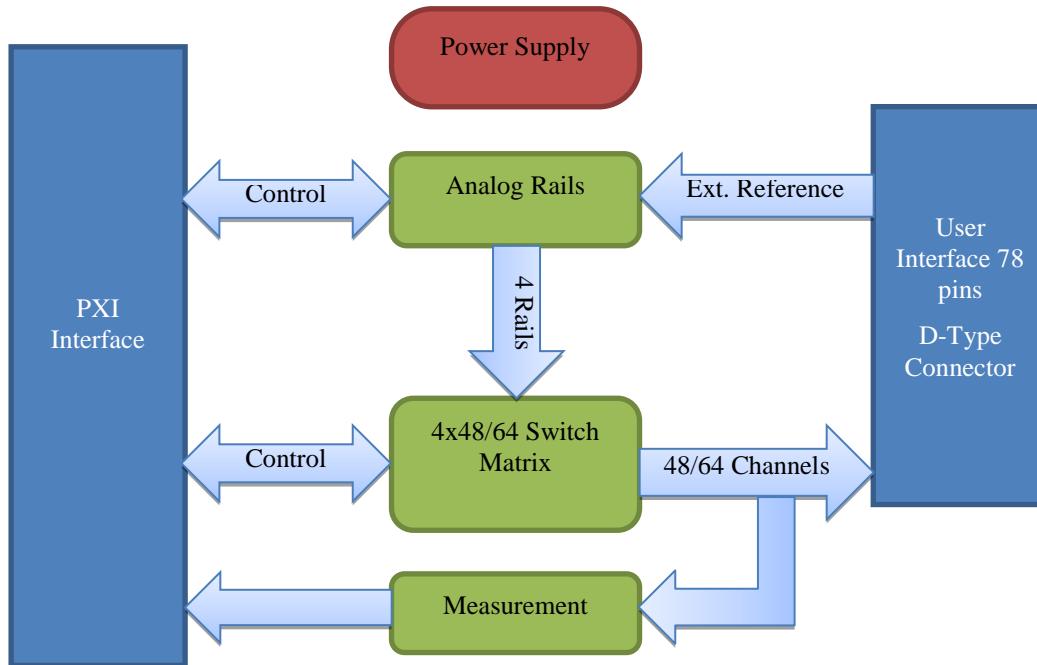


Figure 2-1: GX3348 Board

## Architecture

---

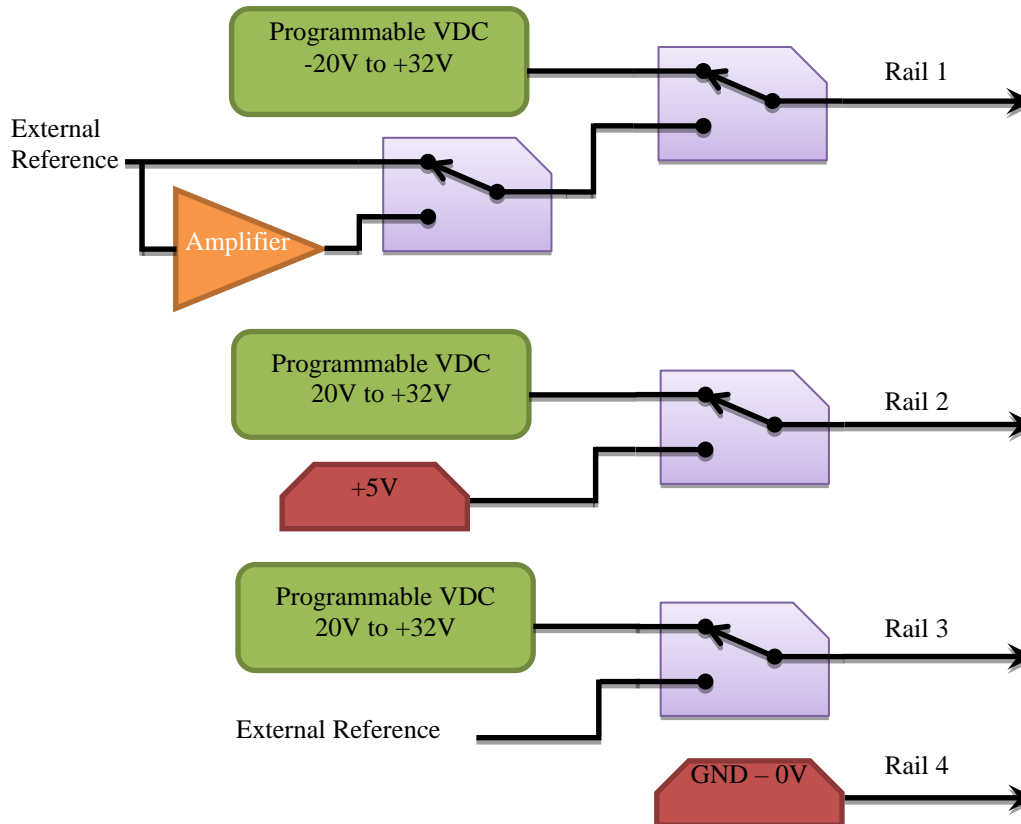
Figure 2-2: GX3348 Block Diagram illustrates the GX3348 architecture. The board is programmed by software using the PXI interface to set the DAC voltages; configure the sources, program the matrix, and measure voltages on the matrix's outputs. The board connector utilizes a 78-pin D-type connector, which provides access to all instrument connections.



**Figure 2-2: GX3348 Block Diagram**

## Output Channels

The GX3348 includes (3) programmable DC sources, a +5VDC source, and an amplifier which can be used to amplify an external input signal. A block diagram of these sources is shown in Figure 2-3. Source selection and routing is done through the software driver.



**Figure 2-3: Analog Rails**

The programmable DC rails have 16-bit resolution and are capable of supplying up to 500 mA. Note that total current from all three sources cannot exceed 500 mA. The +5V source is sourced via the PXI 5 V power supply and total current should not exceed 500 mA. The amplifier has a nominal gain of 6.6 and can amplify signals from DC to > 10 KHz. The output voltage range is -20 V to +32 V @ 500 mA. All rails (A, B and C) are calibrated with a load of 1KOhms over the full range of -20V to 32V.

## Specifications

---

The following table details the specifications of the GX3348:

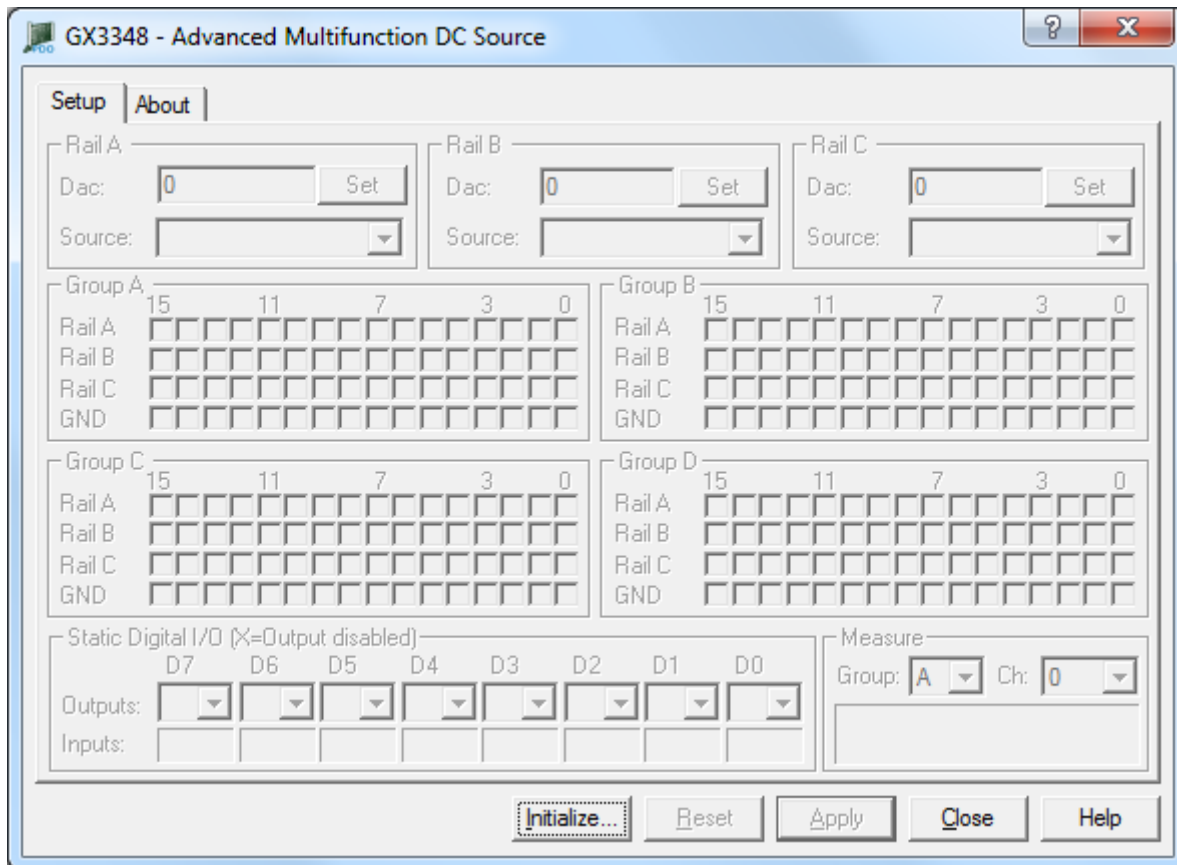
<b>User Channel I/O Specifications</b>	
Channels	48 (GX3348-1), 64(GX3348-2)
Channel Multiplexer Configuration	4 x 48 (GX3348-1), 4 x 64 (GX3348-2)
Maximum Current per Channel	500 mA
Maximum Source Current	500 mA total for all three programmable sources 3 A for all other sources
Number of Voltage Rails	4
Voltage Rails	<ul style="list-style-type: none"> <li>• -20 to +32 VDC or Ext input (with amplifier or bypassed)</li> <li>• -20 to +32 VDC or 5 VDC</li> <li>• -20 to +32 VDC or Ext Input</li> <li>• Ground (0 volts)</li> </ul>
Number of DACs	3
DAC Resolution	16-bit
DAC Output Accuracy	$\pm 1$ LSB, $\pm 10$ mV (with 1000 Ohm load)
Slew Rate (DAC or amplifier)	6 V/ $\mu$ s, 1 K ohm load in parallel with 50 pF
Amplifier Gain	6.6, nominal
Amplifier Bandwidth	DC to 10 KHz
Amplifier Output Swing	-20 to +32 V max.
DAC and Amplifier Output Protection	Short circuit protected to ground
Measurement Function	Measure voltage on all 48 (GX3348-1) or 64(GX3348-2) channels Measurement range: +/- 40 V Resolution: 16-bits Accuracy 5mV
Power On State	All channels / rails open
<b>Physical and Environmental Specifications</b>	
Operating Temperature	0 to+55°C
Storage Temperature	-20 to+70°C
Connector	78 pin, D-sub, female
Size	6U PXI, single slot
Weight	14 oz.



## Virtual Panel Description

The GXPDO includes a virtual panel program, which provides full access to the various configuration settings and operating modes. To understand the front panel operation, it is best to become familiar with the functionality of the board.

To open the virtual panel application, select GX3348 Panel from the Marvin Test Solutions, GXPDO menu under the Start menu. The GX3348 virtual panel opens as shown here:



**Figure 2-4: GX3348 Virtual Panel (not Initialized)**

The functions of the panel bottom controls is shown below:

**Initialize** Opens the Initialize Dialog (see Initialize Dialog paragraph) in order to initialize the board driver. The current settings of the selected counter **will not change after calling initialize**. The panel will reflect the current settings of the counter after the Initialize dialog closes.

**Reset** - resets the PXI board settings to their default state and clears the reading.

**Apply** – applies changed settings to the board

**Close** - closes the panel. Closing the panel **does not affect** the counter settings.

**Help** - opens the on-line help window. In addition to the help menu, the caption shows a **What's This Help** button (?) button. This button can be used to obtain help on any control that is displayed in the panel window. To display the What's This Help information click on the (?) button and then click on the control – a small window will display the information regarding this control.

## Virtual Panel Initialize Dialog

The Initialize dialog initializes the driver for the selected counter board. The counter settings **will not change** after initialize is called. Once initialize, the panel will reflect the current settings of the counter.

The Initialize dialog supports two different device drivers that can be used to access and control the board:

1. **Use HW/PXI Explorer** – this is the device driver installed by the setup program and is the default driver. When selected, the **Slot Number** list displays the available counter boards installed in the system and their slots. The chassis, slots, devices and their resources are also displayed by the HW resource manager, **PXI/PCI Explorer** applet that can be opened from the Windows Control Panel. The PXI/PCI Explorer can be used to configure the system chassis, controllers, slots and devices. The configuration is saved to PXISYS.INI and PXIeSYS.INI located in the Windows folder. These configuration files are also used by VISA. The following figure shows the slot number 0x107 (chassis 1 Slot 6). This is the slot number argument (*nSlot*) passed by the panel when calling the driver **Gx3348Initialize** function used to initialize driver with the specified board.

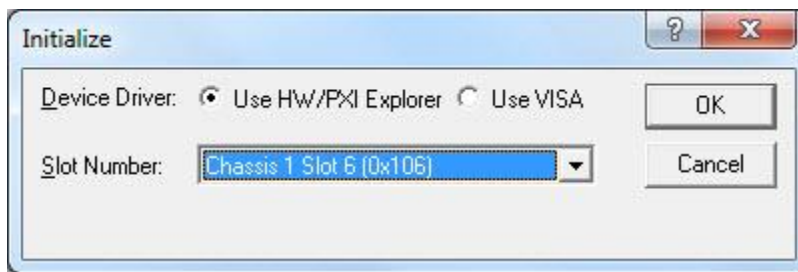


Figure 2-5: Initialize Dialog Box using Marvin Test Solutions' HW driver

2. **Use VISA** – this is a third party device driver usually provided by National Instrument (NI-VISA). When selected, the **Resource** list displays the available boards installed in the system and their VISA resource address. The chassis, slots, devices and their resources are also displayed by the VISA resource manager, **Measurement & Automation** (NI-MAX) and in Marvin Test Solutions **PXI/PCI Explorer**. The following figure shows PXI3::11::INSTR as the VISA resource (PCI bus 3 and Device 11). This is VISA resource string argument (*szVisaResource*) passed by the panel when calling the driver **Gx3348InitializeVisa** function to initialize the driver with the specified board.

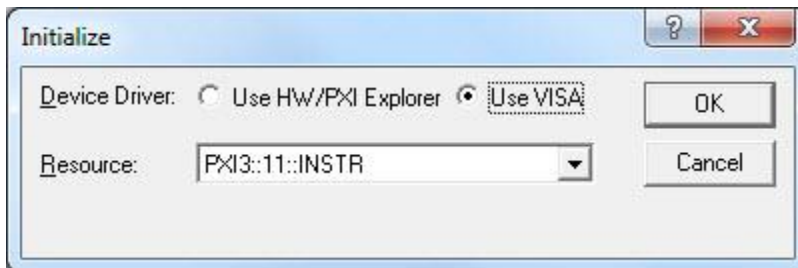
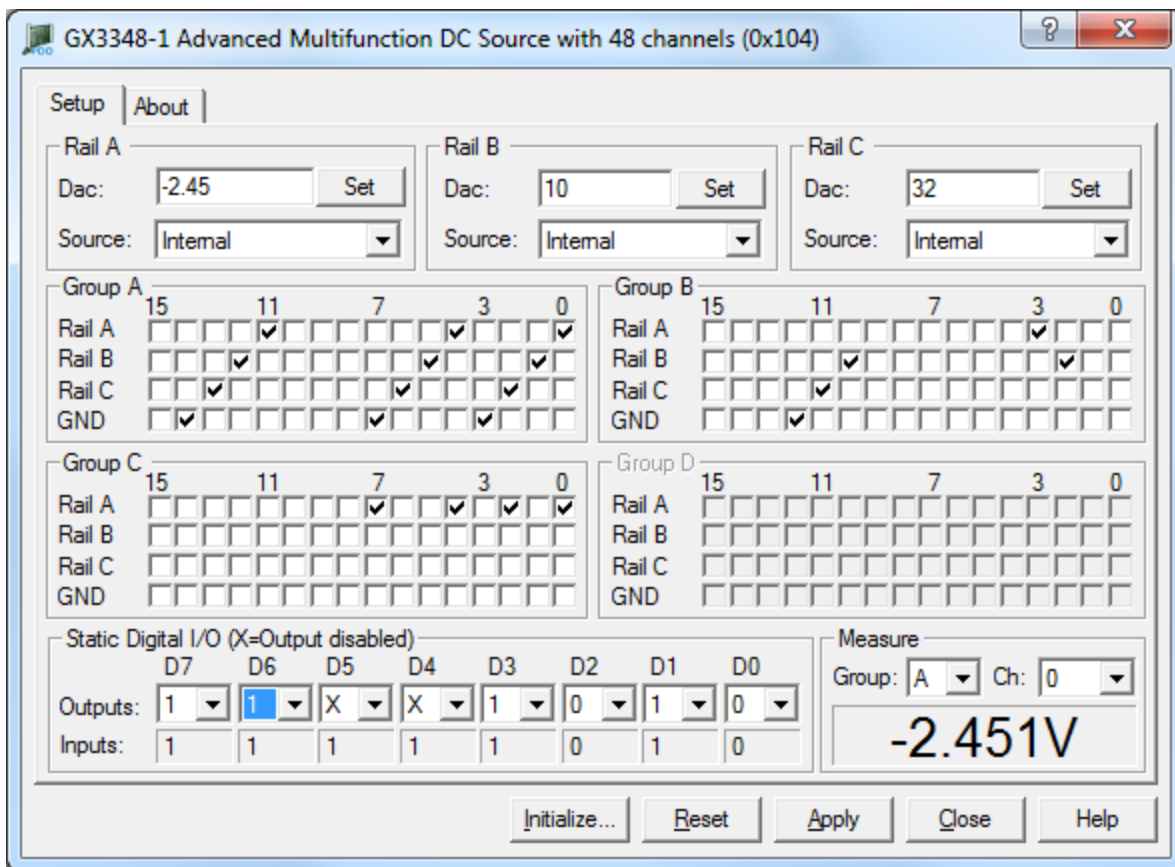


Figure 2-6: Initialize Dialog Box using VISA resources

## Virtual Panel Setup Page

After the board is initialized the panel is enabled and will display the current setting of the board. The panel caption will show the board address (0x107 in this example). The following figure shows the **Setup** page settings:



**Figure 2-7: GX3348 Virtual Panel (Initialized)**

The following controls are shown in the Setup page:

### Rail A Settings Group Box

**A Text Field:** Sets/Displays DAC A Voltage

**Set Buttons:** Set respective DAC's voltage to the value entered in the Text Field.

**Source dropdown list box:** Lists the different Rail A sources: Internal, External, Ext. Amplified.

### Rail B Settings Group Box

**A Text Field:** Sets/Displays DAC B Voltage

**Set Buttons:** Set respective DAC's voltage to the value entered in the Text Field.

**Source dropdown list box:** Lists the different Rail C sources: Internal, Internal 5V.

### Rail C Settings Group Box

**A Text Field:** Sets/Displays DAC C Voltage

**Set Buttons:** Set respective DAC's voltage to the value entered in the Text Field.

**Source dropdown list box:** Lists the different Rail C sources: **Internal, External**

### **Group A-D Group Boxes**

**Check Boxes:** Open/Close relay in matrix for a given Rail (Rail A – Rail D) and Channel (0-15) within a particular Group (A-D). Check indicates closed relay and no check indicates open relay

### **Measure Group Box**

**Group Combo Box:** Select Group to measure

**Channel Combo Box:** Select channel within selected group to measure

**Measurement Display:** Shows the measured voltage at a given Group and Chanel

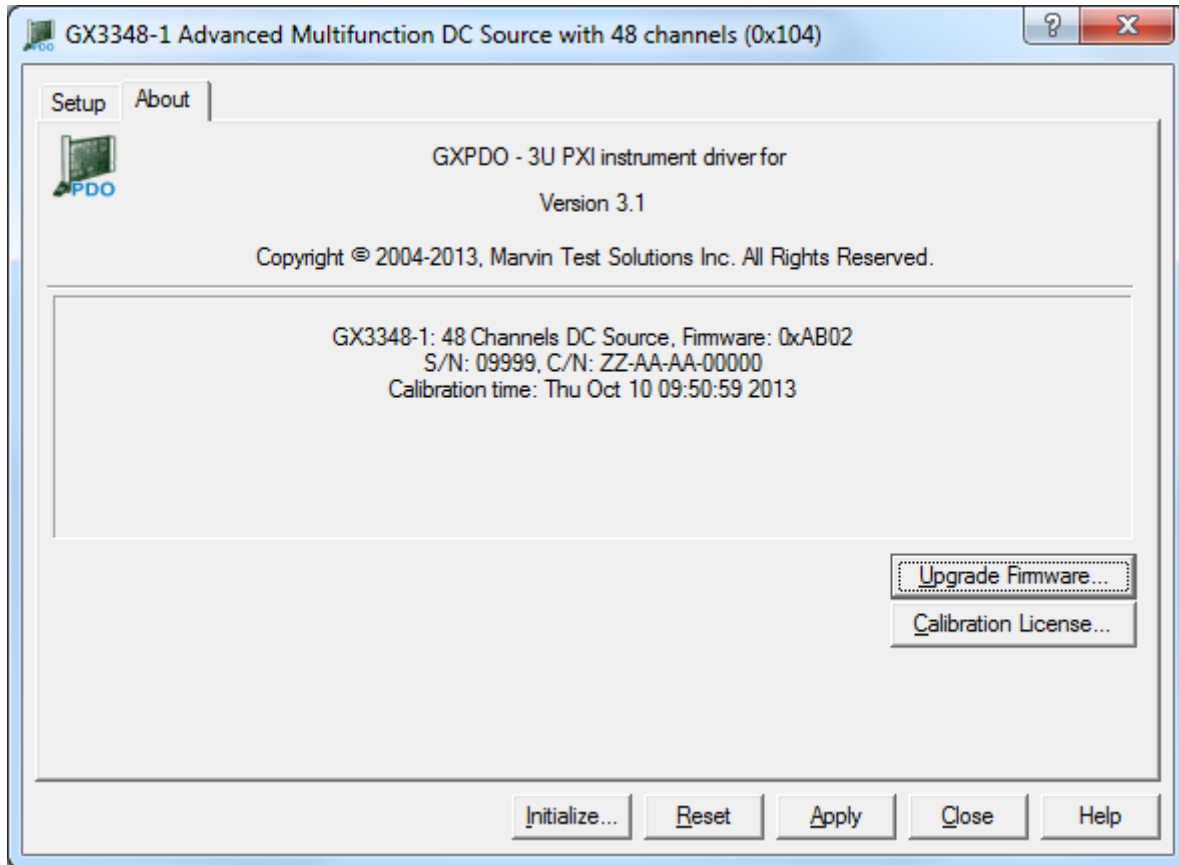
### **Static Digital I/O Group Box**

**Outputs D0 to D7 dropdown list:** Sets/Displays the outputs state and level.

**Inputs D0 to D7 text box:** displays the Digital output lines states.

## Virtual Panel About Page

Clicking on the **About** tab will show the **About** page as shown below:



**Figure 2-8: GX3348 Virtual Panel – About Page**

The following controls are shown in the **About** page:

The top part of the **About** page displays version and copyright of the GXPDO driver. The bottom part displays the board summary. The board summary lists the Boards Type, e.g. GX3348, Serial Number, Control String and Calibration Information.

**Calibration License...** button opens the **License Setup** dialog, see “Chapter 5: Calibration “for details.

The **About** page also contains a button **Upgrade Firmware...** used to upgrade the board Firmware and FPGA. This button maybe used only when the board requires upgrade as directed by Marvin Test Solutions support. The upgrade requires a firmware file (.rpd) that is written to the board FPGA. After the upgrade is complete, you must shut down the computer to recycle power to the board.



## Chapter 3 - Installation and Connections

### Getting Started

---

This section includes general hardware installation procedures for the GX3348 board and installation instructions for the GX3348 - GXPDO software. Before proceeding, please refer to the appropriate chapter to become familiar with the board being installed.

To Find Information on...	Refer to...
GXPDO Software Installation	This Chapter
Hardware/Board Installation	This Chapter
Software Function Reference	Chapter 4
Calibration	Chapter 5
Software Function Reference	Chapter 6

### Packing List

All GX3348 boards have the same basic packing list, which includes:

1. GX3348 Board
2. GXPDO Software CD

### Unpacking and Inspection

After removing the board from the shipping carton:

---



**Caution** - Static sensitive devices are present. Ground yourself to discharge static.

---

1. Remove the board from the static bag by handling only the metal portions.
2. Be sure to check the contents of the shipping carton to verify that all of the items found in it match the packing list.
3. Inspect the board for possible damage. If there is any sign of damage, return the board immediately. Please refer to the warranty information at the beginning of the manual.

### System Requirements

All GX3348 instrument boards are designed to run on PXI compatible computer running Windows (32/64 bit), LabView/Real-Time or Linux.

Each board requires one unoccupied 6U PXI bus slot.

## Installation of the GXPDO Software

---

This section describes installation for Windows, for LabView/Real-Time, see the ReadMe.txt file installed under the product main folder. For Linux installation, see the **GtLinux** software package ReadMe.txt file. Before installing the board, it is recommended to install the software as described in this section:

1. Insert the Marvin Test Solutions CD-ROM and locate the **GXPDO.EXE** setup program. If your computer's Auto Run is configured, when inserting the CD, a browser will show several options, select the Marvin Test Solutions Files option, then locate the setup file. If Auto Run is not configured, you can open the Windows explorer and locate the setup files (usually located under \Files\Setup folder). Alternatively, you can also download and install the latest GXPDO software package from Marvin Test Solutions web site ([www.marvintest.com](http://www.marvintest.com)).
2. Run the setup and follow the instruction on the Setup screen to install the software.

---

**Note:** You may be required to restart the setup after logging-in as a user with Administrator privileges. This is required in-order to upgrade your system with newer Windows components and to install the HW kernel-mode device drivers that are required by the GXPDO driver to access resources on your board.

---

3. The first setup screen to appear is the Welcome screen. Click **Next** to continue.
4. Enter the folder where the software is to be installed. Either click **Browse** to set up a new folder, or click **Next** to accept the default entry of C:\Program Files\Marvin Test Solutions\GXPDO for 32-bit Windows or C:\Program Files (x86)\Marvin Test Solutions\GXPDO for 64-bit Windows.
5. Select the type of Setup you wish and click **Next**. You can choose between **Typical**, **Run-Time** and **Custom** setups. **Typical** setup type installs all files. **Run-Time** setup type will install only the files required for controlling the board either from its driver or from its virtual panel. **Custom** setup type lets you select from the available components.

The program will now start its installation. During the installation, Setup may upgrade some of the Windows shared components and files. The Setup may ask you to reboot after it complete if some of the components it replaced were used by another application during the installation – do so before attempting to use the software.

You can now continue with the installation to install the board. After the board installation is complete, you can test your installation by starting a panel program that let you control the board interactively. The panel program can be started by selecting it from the Start, Programs, GXPDO menu located in the Windows Taskbar.



## Overview of the GXPDO Software

---

Once the software installed, the following tools and software components are available:

- **PXI/PCI Explorer applet** – use to configure the PXI chassis, controllers and devices. This is required for accurate identification of your PXI instruments later on when installed in your system. The applet configuration is saved to PXISYS.ini and PXIE SYS.ini that are used by Marvin Test Solutions instruments, the VISA provider and VISA based instruments drivers. In addition, the applet can be used to assign chassis numbers, Legacy Slot numbers and instruments alias names.

**VISA** is a standard maintained by the VXI Plug & Play System Alliance and the PXI Systems Alliance organizations (<http://www.vxipnp.org/>, <http://www.pxisa.org/>). VISA provides a standard way for instrument manufacturers and users to write and use instruments drivers. The VISA resource managers such as National Instruments **Measurement & Automation** (NI-MAX) can display and configure instruments and their address (similar to Marvin Test Solutions' PXI/PCI Explorer).

- **GXPDO Panel** – use to configure, control and display the board settings.
- **GXPDO driver** - A DLL based function library (GXPDO.DLL or GXPDO64.DLL, located in the Windows System folder) used to program and control the board. The driver uses Marvin Test Solutions' HW driver or VISA supplied by third party vendor to access and control the GXPDO board.
- **Programming files and examples** – interface files and libraries for various programming tools, see later in this chapter for a complete list of files, programming languages and development tools supported by the driver.
- **Documentation** – On-Line help (compiled HTML help file - chm file) and User's Guide (Adobe Acrobat - pdf).

## Configuring Your PXI System using the PXI/PCI Explorer

To configure your PXI/PCI system using the **PXI/PCI Explorer** applet follow these steps:

1. **Start the PXI/PCI Explorer applet.** The applet can be start from the Windows Control Panel or from the Windows Start Menu, **Marvin Test Solutions, HW, PXI/PCI Explorer**.
2. **Identify Chassis and Controllers.** After the PXI/PCI Explorer started, it will scan your system for changes and will display the current configuration. The PXI/PCI Explorer automatically detects systems that have Marvin Test Solutions controllers and chassis. In addition, the applet detects PXI-MXI-3/4 extenders in your system (manufactured by National Instruments). If your chassis is not displayed in the explorer main window, use the Identify Chassis/Controller commands to identify your system. Chassis and Controller manufacturers should provide INI and driver files for their chassis and controllers to be used by these commands.
3. **Change chassis numbers, PXI devices Legacy Slot numbering and PXI devices Alias names.** These are optional steps to be performed if you would like your chassis to have different numbers. Legacy slots numbers are used by older Marvin Test Solutions or VISA drivers. Alias names can provide a way to address a PXI device using your logical name (e.g. "COUNTER1"). For more information regarding these numbers, see the **GxPdoInitialize** and **GxPdoInitializeVisa** functions.
4. **Save you work.** PXI Explorer saves the configuration to the following files located in the Windows folder: PXISYS.ini, PXIeSYS.ini and GxPxiSys.ini. Click on the **Save** button to save you changes. The PXI/Explorer prompt you to save the changes if changes were made or detected (an asterisk sign ' \* ' in the caption indicated changes).

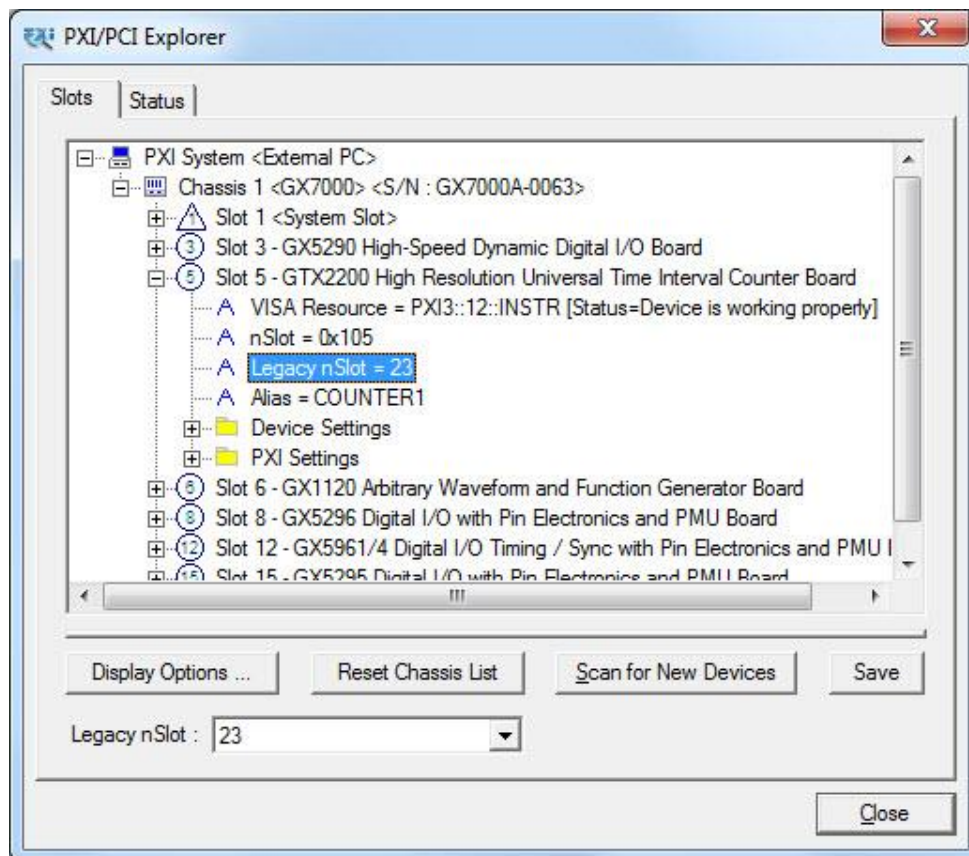


Figure 3-1: PXI/PCI Explorer

## Board Installation

---

### Before you Begin

- Install the software driver as described in the prior section.
- Configure your PXI/PC system using **PXI/PCI Explorer** as described in the prior section.
- Verify that all the components listed in the packing list (see previous paragraph) are present.

### Electric Static Discharge (ESD) Precautions

To reduce the risk of damage to the board, the following precautions should be observed:

- Leave the board in the anti-static bags until installation requires removal. The anti-static bag protects the board from harmful static electricity.
- Save the anti-static bag in case the board is removed from the computer in the future.
- Carefully unpack and install the board. Do not drop or handle the board roughly.
- Handle the board by the edges. Avoid contact with any components on the circuit board.



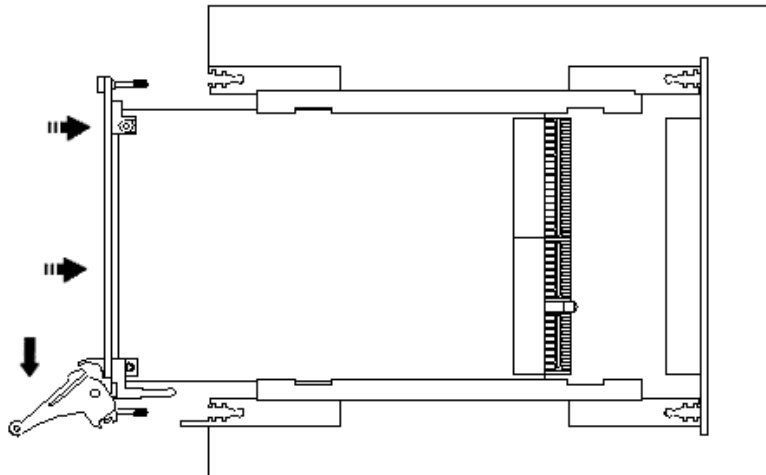
**Caution** - Do not insert or remove any board while the computer is on. Turn off the power from the PXI chassis before installation.

---

### Installing a Board

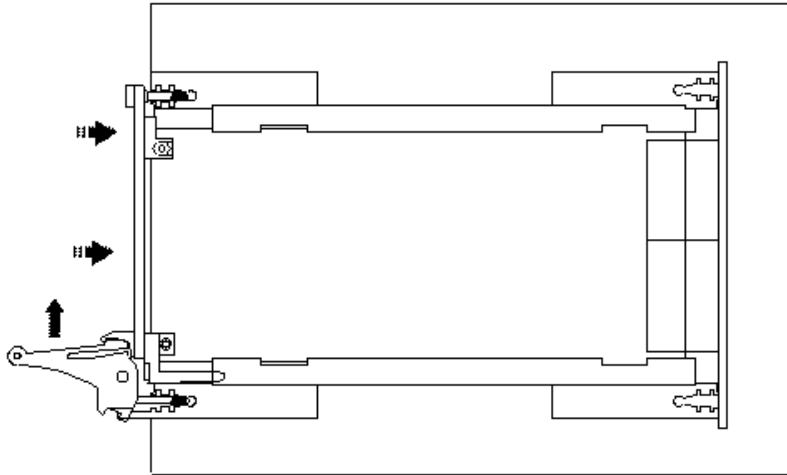
Install the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI empty slot on the PXI chassis.
3. Place the module edges into the PXI chassis rails (top and bottom).
4. Carefully slide the PXI board to the rear of the chassis, make sure that the ejector handles are pushed **out** (as shown in Figure 3-2).



**Figure 3-2: Ejector handles position during module insertion**

- After you feel resistance, push in the ejector handles as shown in Figure 3-3 to secure the module into the frame.



**Figure 3-3: Ejector handles position after module insertion**

- Tighten the module's front panel to the chassis to secure the module in.
- Connect any necessary cables to the board.
- Plug the power cord in and turn on the PXI chassis.

### Plug & Play Driver Installation

Plug & Play operating systems such as Windows notifies the user that a new board was found using the **New Hardware Found** wizard after restarting the system with the new board.

If another Marvin Test Solutions board software package was already installed, Windows will suggest using the driver information file: HW.INF. The file is located in your Program Files folder under \Marvin Test Solutions\HW folder. Click **Next** to confirm and follow the instructions on the screen to complete the driver installation.

If the operating system was unable to find the driver (since the driver was not installed prior to the board installation), you may install the software as described in the prior section, then click on the **Have Disk** button and browse to select the HW.INF file located in your Program File folder under \Marvin Test Solutions\HW.

If you are unable to locate the driver click **Cancel** to the found New Hardware wizard and exit the New Hardware Found Wizard, install the GXPDO driver, reboot your computer and repeat this procedure.

The Windows Device Manager (open from the System applet from the Windows Control Panel) must display the proper board name before continuing to use the board software (no Yellow warning icon shown next to device). If the device is displayed with an error, you can select it and press delete and then press F5 to rescan the system again and to start the New Hardware Found wizard.

## Removing a Board

Remove the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI slot on the PXI chassis.
3. Disconnect and remove any cables/connectors connected to the board.
4. Un-tighten the module's front panel screws to the chassis.
5. Push out the ejector handles and slide the PXI board away from the chassis.
6. Optionally - uninstall the software by running the setup again (or from the Windows Control Pane, Programs and Features or Add Remove Programs applet) and selecting Remove/Uninstall.

## Connectors and Accessories

---

The following accessories are available from Marvin Test Solutions for the GX3348 board.

Part / Model Number	Description
GT97102	3' Harness, 78 pin connector on one end, loose wires (numbered) other end

**Table 3-1: Spare Connectors and Accessories**

## Connectors

---

Table 3-2 shows the GX3348 J6 connector. The GX3348 has one user DB78 connectors. The following tables describes the connectors pinouts.

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	ChA0	21	ChB0	40	ChC0	60	*ChD0
2	ChA1	22	ChB1	41	ChC1	61	*ChD1
3	ChA2	23	ChB2	42	ChC2	62	*ChD2
4	ChA3	24	ChB3	43	ChC3	63	*ChD3
5	ChA4	25	ChB4	44	ChC4	64	*ChD4
6	ChA5	26	ChB5	45	ChC5	65	*ChD5
7	ChA6	27	ChB6	46	ChC6	66	*ChD6
8	ChA7	28	ChB7	47	ChC7	67	*ChD7
9	ChA8	29	ChB8	48	ChC8	68	*ChD8
10	ChA9	30	ChB9	49	ChC9	69	*ChD9
11	ChA10	31	ChB10	50	ChC10	70	*ChD10
12	ChA11	32	ChB11	51	ChC11	71	*ChD11
13	ChA12	33	ChB12	52	ChC12	72	*ChD12
14	ChA13	34	ChB13	53	ChC13	73	*ChD13
15	ChA14	35	ChB14	54	ChC14	74	*ChD14
16	ChA15	36	ChB15	55	ChC15	75	*ChD15
17	External A	37	DIO2	56	External C	76	DIO6
18	DIO0	38	DIO3	57	DIO4	77	DIO7
19	DIO1	39	GND	58	DIO5	78	GND
20	GND			59	GND		

\*ChD0-ChD15 available only on the GX3348

**Table 3-2: Output Connector J6**

## Installation Folders

---

The GXPDO driver files are installed in the default directory C:\Program Files\Marvin Test Solutions\GXPDO. You can change the default GXPDO directory to one of your choosing at the time of installation.

During the installation, GXPDO Setup creates and copies files to the following directories:

Name	Purpose / Contents
...\Marvin Test Solutions\GXPDO	The GXPDO directory. Contains panel programs, programming libraries, interface files and examples, on-line help files and other documentation.
...\Marvin Test Solutions\HW	HW device driver. Provide access to your board hardware resources such as memory, IO ports and PCI board configuration. See the README.TXT located in this directory for more information.
...\ATEasy\Drivers	ATEasy drivers directory. GXPDO Driver and example are copied to this directory only if <i>ATEasy</i> is installed to your machine.
Windows System Folders	Windows System directory. Contains the GXPDO.DLL or GXPDO64.DLL driver, HW driver shared files and some upgraded system components, such as the HTML help viewer, etc.

## GXPDO Driver Files Description

---

The Setup program copies the GXPDO driver, a panel executable, the GXPDO help file, the README.TXT file, and driver samples. The following is a brief description of each installation file:

### Driver File and Virtual Panel

- GXPDO.DLL and GXPDO64.DLL - 32/64 bit Windows DLLs for 32/64 bit applications running under Windows.
- GXPDOPANEL.EXE and GXPDOPANEL64.EXE – An instrument front panel program for all GXPDO supported boards.

### Interface Files

The following GXPDO interface files are used to support the various development tools:

- GXPDO.H - header file for accessing the DLL functions using the C/C++ programming language. The header file compatible with the following 32-bit development tools:
  - Microsoft Visual C++, Microsoft Visual C++ .NET
  - Borland C++
- GXPDO.LIB and GXPDO64.LIB - Import library for GXPDO.DLL and GXPDO64.DLL (used when linking C/C++ application).
- GXPDOBC.LIB - Import library for GXPDO.DLL (used when linking C/C++ Builder application that uses GXPDO.DLL).
- GXPDO.PAS - interface file to support Pascal or Delphi.
- GXPDO.VB - Supports Microsoft Visual Basic .NET.
- GX3348.drv - ATEasy driver file for GX3348.

- GXPDO.llb – LabView library.

### **On-line Help and Manual**

GXPDO.CHM – On-line version of the GXPDO User's Guide. The help file is provided in a Windows Compiled HTML help file (.CHM). The file contains information about the GX3348 board, programming reference and panel operation.

GXPDO.PDF – On line, printable version of the GXPDO User's Guide in Adobe Acrobat format. To view or print the file you must have the reader installed. If not, you can download the Adobe Acrobat reader (free) from <http://www.adobe.com>.

### **ReadMe File**

README.TXT – Contains important last minute information not available when the manual was printed. This text file covers topics such as a list of files required for installation, additional technical notes, and corrections to the GXPDO manuals. You can view and/or print this file using the Windows NOTEPAD.EXE or any other text file editors.

### **Example Programs**

The sample program includes a C/C++ sample compiled with various development tools, Visual Basic example and an ATEasy sample. Other examples may be available for other programming tools.

#### **Microsoft Visual C++ .NET example files:**

- GXPDOExampleC.cpp - Source file
- GXPDOExampleC.ico - Icon file
- GXPDOExampleC.rc - Resource file
- GXPDOExampleC.vcproj - VC++ .NET project file
- GXPDOExampleC.exe - 32-bit example executable
- GXPDOExampleC64.exe - 64-bit example executable

#### **Embarcadero/Borland C++ Builder example files:**

- GXPDOExampleC.cpp - Source file
- GXPDOExampleC.ico - Icon file
- GXPDOExampleC.rc - Resource file
- GXPDOExampleC.bpr - Borland project file
- GXPDOExampleC.exe - Example executable

#### **Microsoft Visual Basic .NET example files:**

- GXPDOExampleVB.vb - Example form.
- GXPDOExampleVB.resx - Example form resource.
- GXPDOExampleVBApp.config - Example application configuration file.
- GXPDOExampleVBAssemblyInfo.vb - Example application assembly file
- GXPDOExampleVB.vbproj - Project file
- GXPDOExampleVB.exe - Example executable

#### **Microsoft Visual C# .NET example files:**

- GXPDOExampleCS.cs - Example form.
- GXPDOExampleCS.csproj - Project file



- GXPDOExampleCS.exe - Example executable

**ATEasy driver and examples files (ATEasy Drivers directory):**

- GX3348.prj - example project
- GX3348.sys - example system
- GX3348.prg - example program
- GX3348.prj - example project
- GX3348.sys - example system
- GX3348.prg - example program

**Setup Maintenance Program**

---

You can run the Setup again after GXPDO has been installed from the original disk or from the Windows Control Panel – Add Remove Programs applet. Setup will be in the Maintenance mode when running for the second time. The Maintenance window show below allows you to modify the current GXPDO installation. The following options are available in Maintenance mode:

- **Modify.** When you want to add or remove GXPDO components.
- **Repair.** When you have corrupted files and need to reinstall.
- **Remove.** When you want to completely remove GXPDO.

Select one of the options and click **Next**.

Follow the instruction on the screen until Setup is complete.



## Chapter 4 - Programming the Board

This chapter contains information about how to program the GX3348 using the GXPDO driver. The driver contains functions to initialize, reset, and control the PXI board. This chapter includes a brief description of the functions, as well as how and when to use them, with code examples. Chapter 5 contains a complete and detailed description of the available programming functions.

The driver supports many development tools. This chapter describes using these tools with the driver. The GXPDO installation directory contains examples written for these development tools. Chapter 3 lists the available examples.

An example using the DLL driver with Microsoft Visual C++ is at the end of this chapter. The driver functions and parameters are identical for all operating systems and development tools, and this example serves as a guide for use other programming languages.

---

### The GXPDO Driver

The driver is a 32-bit Windows DLL file: GXPDO.DLL and 64-bit DLL: GXPDO64.DLL. The DLL is used with 32-bit or a 64-bit applications running under Windows. The HW device driver is installed by the setup program and is shared by other Marvin Test Solutions products (ATEasy, GXPIO, GXSW, etc.). The DLLs can also use VISA (provided by a third party) to access the board hardware instead of the provided HW driver.

The DLL can be used with various development tools such as Visual C++, C++ Builder, Microsoft Visual Basic, Pascal or Delphi, ATEasy, LabView and more. The following paragraphs describe how to create an application that uses the driver with various development tools. Refer to the paragraph describing the specific development tool for more information.

---

### Programming Using C/C++ Tools

The following steps are required to use the GXPDO driver with C/C++ development tools:

- Include the GXPDO.H header file in the C/C++ source file that uses the GX3348 function. The file contains function prototypes and constant declarations to be use by the C/C++ compilers for your application.
- Add the required .LIB file to the projects. This can be the import library GXPDO.LIB for Microsoft Visual C++ for 32-bit applications (or other 32-bit compiler such as LabWindows/CVI), GXPDO64.LIB for 64-bit applications, and GXPDOBC.LIB for C++ Builder Windows- based applications that explicitly load the DLL by calling the Windows **LoadLibrary** API should not include the .LIB file in the project.
- Add code to call the Gx3348 as required by the application.
- Build the project.
- Run, test, and debug the application.

---

### Programming Using Visual Basic

To use the driver with Microsoft Visual Basic .NET developers must use the GXPDO.VB.

The file can be loaded within the integrated development environment (IDE) using *Add File* from the Visual Basic IDE *File menu*. The GXPDO.VB contains function declarations for the DLL driver.

---

### Programming Using C#

To use the driver with Microsoft Visual C# .NET developers must use the GXPDO.cs.

The file can be loaded within the integrated development environment (IDE) using *Add File* from the Visual Basic IDE *File menu*. The GXPDO.cs contains function declarations for the DLL driver.

## Programming Using Pascal/Delphi

---

To use the driver with Borland Pascal or Delphi, the user must include the GXPDO.PAS to the project. The GXPDO.PAS file contains a **unit** with function prototypes for the DLL functions. Include the GX3348 unit in the **uses** statement before making calls to the GX3348 functions.

## Programming Using ATEasy®

---

The GXPDO package provides an ATEasy driver for the PXI board. The ATEasy driver uses the same GXPDO.DLL to program the board for all development environments. An example project that contains a program and a system file pre-configured with the ATEasy driver is also provided. Use the property dialog to change the driver shortcut property in the System Drivers sub-module and change the PCI slot number to reflect your current installation before running the example.

The ATEasy driver plain-languages commands are easier to use than using the DLL functions directly. The driver commands will generate an exception if the function call fails. That allows the ATEasy application to trap errors without directly checking the status code returned by the DLL function. The driver commands check the status parameter after each function call.

The ATEasy driver contains commands that are similar to the DLL functions in name and parameters, with the following exceptions:

- The driver handles the *nHandle* parameter automatically. ATEasy handles board instances as driver logical names i.e. GXPDO1, GXPDO2 for GX3348. Direct access to the parameter is possible using the property operator (the “dot” operator) once the parameter is set public in the ATEasy driver.
- The driver handles the *nStatus* parameter automatically. The Get Status command in the ATEasy Driver provides access to the board’s status. After calling a DLL function, the ATEasy driver will check the returned status and will call the **CheckError** procedure. An error status generates an exception that can be easily trapped by the application using the **OnError** module event or using the **try-catch** statement. ATEasy will notify the user or developer with an error dialog if the condition is not handled by other ATEasy code. Direct access to the parameter is possible using the property operator (the “dot” operator) once the parameter is set public in the ATEasy driver.

Some ATEasy drivers contain additional commands to permit easier access to the board features. For example, parameters for a function may be omitted by using a command item instead of typing the parameter value. The plain-language commands are self-documenting. Their syntax is similar to an English language statement. In addition, you can use the commands from the code editor context menu or by using the ATEasy’s code completion feature instead of typing them directly.

## Programming Using LabView and LabView/Real Time

---

To use the driver with LabView use the provided lab view library GXPDO.llb. The library is located in the GXPDO folder. An example for LabView is also provided in the Examples folder. A DLL located in the LabViewRT folder can be used for deployment with LabView/Real-Time.

## Using and Programming under Linux

---

Marvin Test Solutions provides a separate software package, **GtLinux**, with a Linux driver (Marvin Test Solutions Drivers Pack for Linux). The software package can be download from the Marvin Test Solutions website. See the ReadMe.txt in that package for more information regarding using and programming the driver under Linux.

## Using the GXPDO Driver Functions

The GXPDO driver contains a set of functions for the GX3348. Functions names that starts with the **GXPDO** prefix applies to all GXPDO boards (i.e. **Gx3348Reset**). The GXPDO functions are designed with consistent set of arguments and functionality. All boards have a function that initializes the GXPDO driver for a specific board, reset the board, and display the virtual panel. All the functions use handles to identify and reference a specific board and all functions return status and share the same functions to handle error codes.

### Initialization, HW Slot Numbers and VISA Resource

The GXPDO driver supports two device drivers HW and VISA, which are used to initialize, identify and control the board. The user can use the **Gx3348Initialize** to initialize the board 's driver using HW and **Gx3348InitializeVisa** to initialize using VISA. The following describes the two different methods used:

1. **Marvin Test Solutions' HW** - the default device driver that is installed by the GXPDO driver. To initialize and control the board using the HW use the **Gx3348Initialize(*nSlot*, *pnHandle*, *pnStatus*)** function. The function initializes the driver for the board at the specified PXI slot number (*nSlot*) and returns a board handle. The **PXI/PCI Explorer** applet in the Windows Control Panel displays the PXI slot assignments. You can specify the *nSlot* parameter in the following way:
  - A combination of chassis number (chassis # x 256) with the chassis slot number, e.g. 0x105 for chassis 1 and slot 5. Chassis number can be set by the **PXI/PCI Explorer** applet.
  - Legacy *nSlot* as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet: 23 in this example.

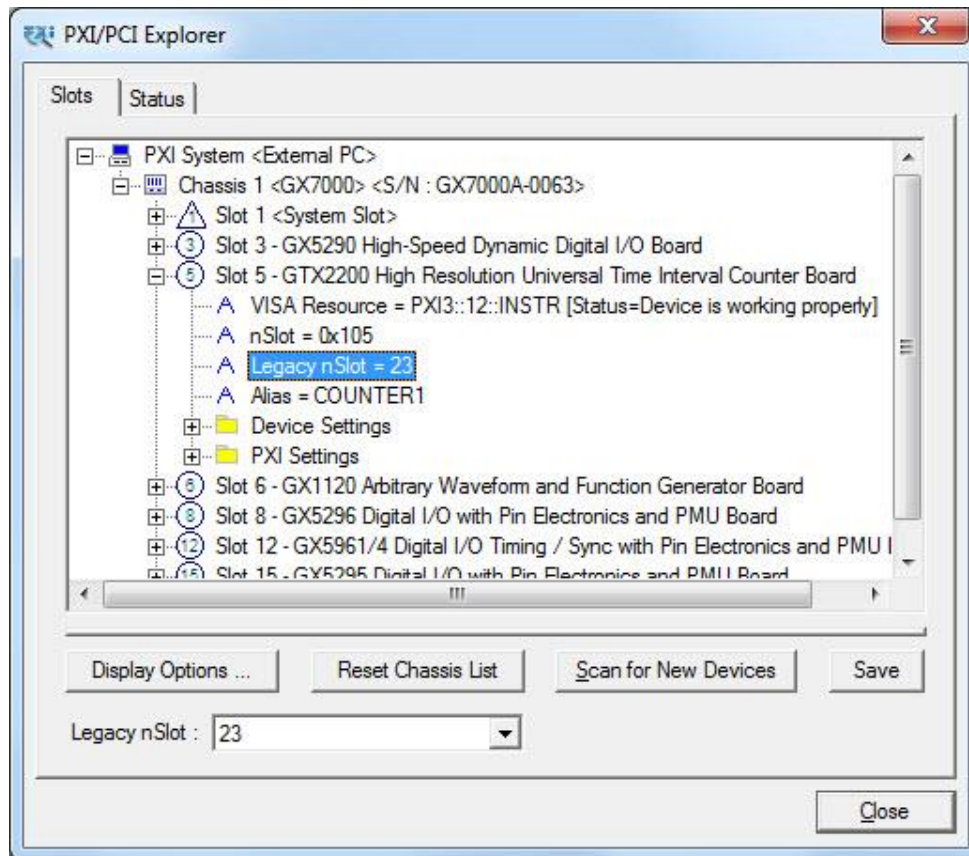


Figure 4-1: PXI/PCI Explorer

2. **VISA** – this is a third party library usually by National Instruments (NI-VISA). You must ensure that the VISA installed supports PXI and PCI devices (not all VISA providers supports PXI/PCI). GXPDO setup installs a VISA compatible driver for the GXPDO board in-order to be recognized by the VISA provider. Use the GXPDO function **Gx3348InitializeVisa** (*szVisaResource*, *pnHandle*, *pnStatus*) to initialize the driver board using VISA. The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as **NI Measurement and Automation (NI\_MAX)**. It is also displayed by Marvin Test Solutions **PXI/PCI Explorer** as shown in the prior figure. The VISA resource string can be specified in several ways as the following examples:

- Using chassis, slot: “PXI0::CHASSIS1::SLOT5”
- Using the PCI Bus/Device combination: “PXI9::13::INSTR” (bus 9, device 9).
- Using alias: “COUNTER1”. Use the PXI/PCI Explorer to set the device alias.

Information about VISA is available at <http://www.pxisa.org>.

The **Gx3348Initialize** function returns a handle that is required with other driver functions to program the board. This handle is usually saved in the program in a global variable for later use when calling other functions. The initialize function does not change the state of the board or its settings.

### Board Handle

The board handle argument, *nHandle*, passed (by reference) to the parameter *pnHandle* of the **Gx3348Initialize** or the **Gx3348InitializeVisa** functions is a short integer (16 bits) number. It is used by the GXPDO driver functions to identify the board being accessed by the application. Since the driver supports many boards at the same time, the *nHandle* argument is required to uniquely identify which board is being programmed.

The *nHandle* is created when the application calls the **Gx3348Initialize** function. There is no need to destroy the handle. Calling **Gx3348Initialize** with the same slot number will return the same handle.

Once the board is initialized the handle can be used with other functions to program the board.

### Reset

The Reset function **Gx3348Reset**(*nHandle*, *nStatus*), causes the driver to change all settings to their default state. See the Function Reference for more information regarding the specific board.

### Error Handling

All GXPDO functions pass a fail or success status - *pnStatus* - in the last parameter. A successful function call passes zero in the status parameter upon return. If the status is non-zero, then the function call fails. This parameter can be later used for error handling. When the status is error, the program can call the **GxPdoGetErrorString** function to return a string representing the error. The **GxPdoGetErrorString** reference contains possible error numbers and their associated error strings.

### Driver Version

The **GxPdoGetDriverSummary** function can be used to return the current GXPDO driver version. It can be used to differentiate between the driver versions. See the Function Reference for more information.

### Panel

Calling the **Gx3348Panel** will display the instrument's front panel dialog window. The panel can be used to initialize and control the board interactively. The panel function may be used by the application to allow the user to directly interact with the board.

The **Gx3348Panel** function is also used by the GXPDOPANEL.EXE. or the GXPDOPANEL64.EXE applications that are supplied with this package and provides a stand-alone Windows application that displays the instrument panel.

## Distributing the Driver

---

Once the application is developed, the driver files (GXPDO.DLL or GXPDO64.DLL and the HW device driver files located in the HW folder) can be shipped with the application. Typically, the DLLs should be copied to the Windows System directory. The HW device driver files should be installed using a special setup program HWSETUP.EXE that is provided with GXPDO driver files. Alternatively, you can provide the GXPDO disk to be installed along with the board.

## Sample Programs

---

The following example demonstrates how to program the board using the C programming language under Windows. The example shows how to apply certain voltage to a specified rail.

To run, enter the following command line:

```
GxPdoExample <PciSlot> <operation> <channel | dac | rail> <rail_number | voltage | rail_source>
```

Where:

<PciSlot>	PCI/PXI Explorer slot number where the board reside.
<operation>	Operation code: SCR=Set Channel Rail GCR=Get Channel Rail SDV=Set Dac Voltage GDV=Get Dac Voltage SRS=Set Rail Source
< channel   dac   rail >	Depends on the operation: Channel number: 0-7 for SCR/GCR operations DAC number: 0-2 for SDV/GDV operations Rail number: 0-2 for SRS/GRS operations
<rail_numbe voltage rail_source>	Depends on operation: Rail_number: 0 to 2 Voltage: -10V to 32V Rail_source: 0 to 4

## Sample Program Listing

---

```

/*****
FILE           : GxPdoExampleC.cpp

PURPOSE       : WIN32/LINUX example program for GX1838/GX3348
                boards using the GXPDO driver.

CREATED       : May 2002

COPYRIGHT     : Copyright 2002-2016, Marvin Test Solutions, Inc.

COMMENTS      :

To compile the example:

1. Microsoft VC++
   Load GxPdoExampleC.dsp, .vcproj or .mak, depends on
   the VC++ version from the Project\File/Open... menu
   Select Project/Rebuild all from the menu

2. Borland C++ Builder
   Load GxPdoExampleC.bpr from the Project/Open
   Project... menu
   Select Project/Build all from the menu

3. Linux (GCC for CPP and Make must be available)
   make -fGxPdoExampleC.mk [CFG=Release[64] | Debug[64]] [rebuild |
   clean]

*****/
#ifdef __GNUC__
#include "windows.h"
#endif
#include "GxPdo.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#ifdef __BORLANDC__
#pragma hdrstop
#include <condefs.h>
USELIB("GxPdoBC.lib");
USERC("GxPdoExampleC.rc");
#endif

/*****
//          DisplayMsg
/*****
void DisplayMsg(PCSTR lpszMsg)
{
#ifdef __GNUC__
    MessageBeep(0);
    MessageBox(0, lpszMsg, "GxPdo example program", MB_OK);
#else

```



```

    printf("\r\nGxPdo example program: %s\r\n", lpszMsg);
#endif
    return;
}

//*****
//      __strupr
//*****
char * __strupr(char * sz)
{
    int i;

    for (i=0; sz[i]; i++)
        sz[i]=toupper(sz[i]);
    return sz;
}

//*****
//      DisplayUsage
//*****
void DisplayUsage(void)
{
    DisplayMsg(
        "\r\nThis example shows how to use the GX1838/Gx3348:\r\n\r\n"

        "Usage:\r\n"
        "For 1838:\r\n"
        "GxPdoExampleC 1838 <slot_number> <operation> <channel|dac|"
        "    <rail> <rail_number|voltage|rail_source>"
        "\r\n\r\nWhere : \r\n"
        "<model> - Model of board, 1838\r\n"
        "<slot_number> - PCI/PXI slot number as shown by the PXI "
        "explorer\r\n"
        "<operation> - one of the followings :\r\n"
        "\tSCR=Set Channel Rail\r\n"
        "\tGCR=Get Channel Rail\r\n"
        "\tSDV=Set DAC Voltage\r\n"
        "\tGDV=Get DAC Voltage\r\n"
        "\tSRS=Set Rail Source\r\n"
        "\tGRS=Get Rail Source\r\n\r\n"
        "<channel|dac|rail numbers> - depends on the operation:\r\n"
        "\tchannel number:\t0-7 for SCR/GCR operations\r\n"
        "\tdac number :   \t0-2 for SDV/GDV operations\r\n"
        "\trail number :   \t0-2 for SRS/GRS operations\r\n\r\n"
        "For 3348:\r\n"
        "GxPdoExampleC 3348 <slot_number> <operation> <group|dac>
<channel>"
        "    <rail> <state|voltage>\r\n"
        "\r\n\r\nWhere : \r\n"
        "<model> - Model of board, 3348\r\n"
        "<slot_number> - PCI/PXI slot number as shown by the PXI "
        "explorer\r\n"
        "<operation> - one of the followings :\r\n"
        "\tSCR=Set Channel Rail\r\n"
        "\tGCR=Get Channel Rail\r\n"
        "\tSDV=Set DAC Voltage\r\n"
        "\tGDV=Get DAC Voltage\r\n"

```

```

        "\tSRS=Set Rail Source\r\n"
        "\tGRS=Get Rail Source\r\n\r\n"
        "\tMES=Measure channel voltage\r\n"
        "<group|dac> - channel group or dac number (0-3)\r\n"
        "<channel> - channel to set or measure within a group(0-15)\r\n"
        "<dac> - dac to set (0-2)\r\n"
        "<voltage> - depends on the operation:\r\n"
        "\tSDV : 0-3    \tDAC Voltage (-20 to 30V)\r\n"
        "\r\nTo change command line under Windows:\r\n"
        "\tRight click on the example shortcut from the start menu\r\n"
        "\tand type the new command line\r\n"
    );
    exit(1);
}

//*****
//          CheckStatus
//*****
void CheckStatus(SHORT nStatus)
{
    CHAR    sz[512];

    if (!nStatus) return;
    GxPdoGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    DisplayMsg(sz);
    DisplayMsg("Aborting the program...");
    exit(nStatus);
}

//*****
//          MAIN
//
// This main function receives five parameters
//
// GXPDO model (e.g. 1838 or 3348)
// GXPDO board slot number (e.g. 1)
// GX1838 operations (e.g. SCR=Set Channel Rail,
//                      GCR=Get Channel Rail,
//                      SDV=Set Dac Voltage,
//                      GDV=Get Dac Voltage,
//                      SRS=Set Rail Source,
//                      GRS=Get Rail Source
// GX1838 channel dac rail number depends on operation
//                      0 to 7 for channel
//                      0 to 2 for dac
//                      0 to 2 for rail
// rail_number voltage rail_source depends on operation
//                      0 to 2 for rail_number
//                      -10V to 32V for voltage
//                      0 to 4 for rail_source
// GX3348 operations (e.g. SDV=Set Dac Voltage,
//                      SCR=Set Channel Rail,
//                      GCR=Get Channel Rail,
//                      SDV=Set Dac Voltage,
//                      GDV=Get Dac Voltage,
//                      SRS=Set Rail Source,
//                      GRS=Get Rail Source

```

```

//                                     MES=Measure channel voltage
// GX3348 group, channel, rail
//                                     0 to 3 for group
//                                     0 to 15 for channel
//                                     0 to 2 for dac
//                                     0 to 3 for rail
//*****
int main(int argc, char **argv)
{
    short nBoardType;           // Board model
    short nSlotNum;            // Board slot number
    char* sOperation;          // Board Operation
    short nChannelOrDacOrRail; // Channel or DAC or Rail number
    short nHandle;             // Board handle
    short nStatus;             // Returned status
    short nRailOrRailSource;   // Rail number or rail source
    double dVoltage;           // Dac voltage
    short nGroup;              // Group number
    short nChannel;            // Channel number
    short nDac;                // Dac number
    short nRail;               // Rail number
    short nRailSource;         // rail source

    // Check number of arguments recived
    if (argc<4) DisplayUsage();

    // Parse command line parameters
    nBoardType=(SHORT)strtol(*(++argv), NULL, 0);
    nSlotNum=(SHORT)strtol(*(++argv), NULL, 0);
    sOperation=__strupr(*(++argv));

    if (nSlotNum<0)
        DisplayUsage();

    switch(nBoardType)
    {
        case 1838:
            nChannelOrDacOrRail=(SHORT)strtol(*(++argv), NULL, 0);
            Gx1838Initialize(nSlotNum, &nHandle, &nStatus);
            CheckStatus(nStatus);

            if(!strcmp(sOperation, "SCR"))
            {
                // set channel rail
                if (argc<5) DisplayUsage();
                nRailOrRailSource=(SHORT)strtol(*(++argv), NULL, 0);
                Gx1838SetChannelRail(nHandle, nChannelOrDacOrRail,
                    nRailOrRailSource, &nStatus);
                CheckStatus(nStatus);
                printf("Set Channel %i rail to %i.\n",
                    nChannelOrDacOrRail, nRailOrRailSource);
            }
            else if(!strcmp(sOperation, "GCR"))
            {
                // get channel rail
                Gx1838GetChannelRail(nHandle, nChannelOrDacOrRail,
                    &nRailOrRailSource, &nStatus);
                CheckStatus(nStatus);
                printf("Channel %i rail is %i.\n",

```

```

        nChannelOrDacOrRail, nRailOrRailSource);
    }
else if(!strcmp(sOperation, "SDV"))
{
    // set dac voltage
    if (argc<5) DisplayUsage();
    dVoltage=strtod(*(++argv), NULL);
    Gx1838SetDacVoltage(nHandle, nChannelOrDacOrRail,
        dVoltage, &nStatus);
    CheckStatus(nStatus);
    printf("Set DAC %i voltage to %f.\n",
        nChannelOrDacOrRail, dVoltage);
}
else if(!strcmp(sOperation, "GDV"))
{
    // get dac voltage
    Gx1838GetDacVoltage(nHandle,
        nChannelOrDacOrRail, &dVoltage, &nStatus);
    CheckStatus(nStatus);
    printf("Dac %i voltage is %f.\n",
        nChannelOrDacOrRail, dVoltage);
}
else if(!strcmp(sOperation, "SRS"))
{
    // set rail source
    if (argc<5) DisplayUsage();
    nRailOrRailSource=(SHORT)strtol(*(++argv), NULL, 0);
    Gx1838SetRailSource(nHandle, nChannelOrDacOrRail,
        nRailOrRailSource, &nStatus);
    CheckStatus(nStatus);
    printf("Set rail %i source to %i.\n",
        nChannelOrDacOrRail, nRailOrRailSource);
}
else if(!strcmp(sOperation, "GRS"))
{
    // get rail source
    Gx1838GetRailSource(nHandle, nChannelOrDacOrRail,
        &nRailOrRailSource, &nStatus);
    CheckStatus(nStatus);
    printf("Rail %i source is %i.\n",
        nChannelOrDacOrRail, nRailOrRailSource);
}
break;
//*****
// 3348
//*****
case 3348:
    Gx3348Initialize(nSlotNum, &nHandle, &nStatus);
    CheckStatus(nStatus);
    if(!strcmp(sOperation, "SCR"))
    {
        if (argc<7) DisplayUsage();
        // Set Channel Rail
        nGroup=(SHORT)strtol(*(++argv), NULL, 0);
        nChannel=(SHORT)strtol(*(++argv), NULL, 0);
        nRail=(SHORT)strtol(*(++argv), NULL, 0);

        Gx3348SetChannelRail(nHandle, nGroup, nChannel,
            nRail, &nStatus);
        CheckStatus(nStatus);
    }
}

```

```

else if(!strcmp(sOperation, "GCR"))
{
    if (argc<6) DisplayUsage();
    //Get Channel Rail
    nGroup=(SHORT)strtol(++argv, NULL, 0);
    nChannel=(SHORT)strtol(++argv, NULL, 0);

    Gx3348GetChannelRail(nHandle, nGroup, nChannel,
        &nRail, &nStatus);
    CheckStatus(nStatus);

    if(nRail>=GX3348_RAIL_A && nRail<=GX3348_RAIL_C)
        printf("Channel %d in Group %d connected "
            "to Rail %d", nChannel, nGroup, nRail);
    else if(nRail==GX3348_RAIL_GROUND)
        printf("Channel %d in Group %d connected "
            "to Ground Rail %d", nChannel, nGroup);
    else
        printf("Channel %d in Group %d "
            "disconnected from all rails", nChannel,
            nGroup);
}

else if(!strcmp(sOperation, "SDV"))
{
    if (argc<6) DisplayUsage();
    //Set Dac Voltage
    nDac=(SHORT)strtol(++argv, NULL, 0);
    dVoltage=strtod(++argv, NULL);

    Gx3348SetDac(nHandle, nDac, dVoltage, &nStatus);
    CheckStatus(nStatus);
    printf("DAC %i voltage is %f.\n", nDac, dVoltage);
}

else if(!strcmp(sOperation, "GDV"))
{
    if (argc<5) DisplayUsage();
    //Get Dac Voltage
    nDac=(SHORT)strtol(++argv, NULL, 0);

    Gx3348GetDac(nHandle, nDac, &dVoltage, &nStatus);
    CheckStatus(nStatus);
    printf("DAC %i voltage is %f.\n", nDac, dVoltage);
}

else if(!strcmp(sOperation, "SRS"))
{
    // set rail source
    if (argc<5) DisplayUsage();
    nRail=(SHORT)strtol(++argv, NULL, 0);
    nRailSource=(SHORT)strtol(++argv, NULL, 0);
    Gx3348SetRailSource(nHandle, nRail,
        nRailSource, &nStatus);
    CheckStatus(nStatus);
    printf("Set rail %i source to %i.\n",
        nRail, nRailSource);
}

else if(!strcmp(sOperation, "GRS"))
{
    // get rail source

```

```

        nRail=(SHORT)strtol(++argv, NULL, 0);
        Gx3348GetRailSource(nHandle, nRail,
            &nRailSource, &nStatus);
        CheckStatus(nStatus);
        //printf("Rail %i source is %i.\n",
        //      nRail, nRailSource);
    }
    else if(!strcmp(sOperation, "MES"))
    {
        if (argc<6) DisplayUsage();
        //Measure channel voltage
        nGroup=(SHORT)strtol(++argv, NULL, 0);
        nChannel=(SHORT)strtol(++argv, NULL, 0);
        Gx3348Measure(nHandle, nGroup, nChannel, &dVoltage,
            &nStatus);
        CheckStatus(nStatus);
        printf("Measured voltage on Channel %d in Group %d "
            "is %fV.\n", nChannel, nGroup, dVoltage);
    }
    break;
default:
    DisplayUsage();
}

return 0;
}

//*****
//      End Of File
//*****

```

## Chapter 5 - Calibration

### Introduction

The GX3348 can be calibrated using the GXPDO driver API or by using Marvin Test Solutions's **CalEasy** software. This chapter focuses on using the driver API for calibration. Using either method of calibration requires purchasing of a calibration license for CalEasy or for the API from Marvin Test Solutions.

The GXPDO driver API exposes a set of functions to allow the end user to create their own calibration program. The GX3348 stores all calibration data within an onboard EEPROM.

### Hardware Requirements

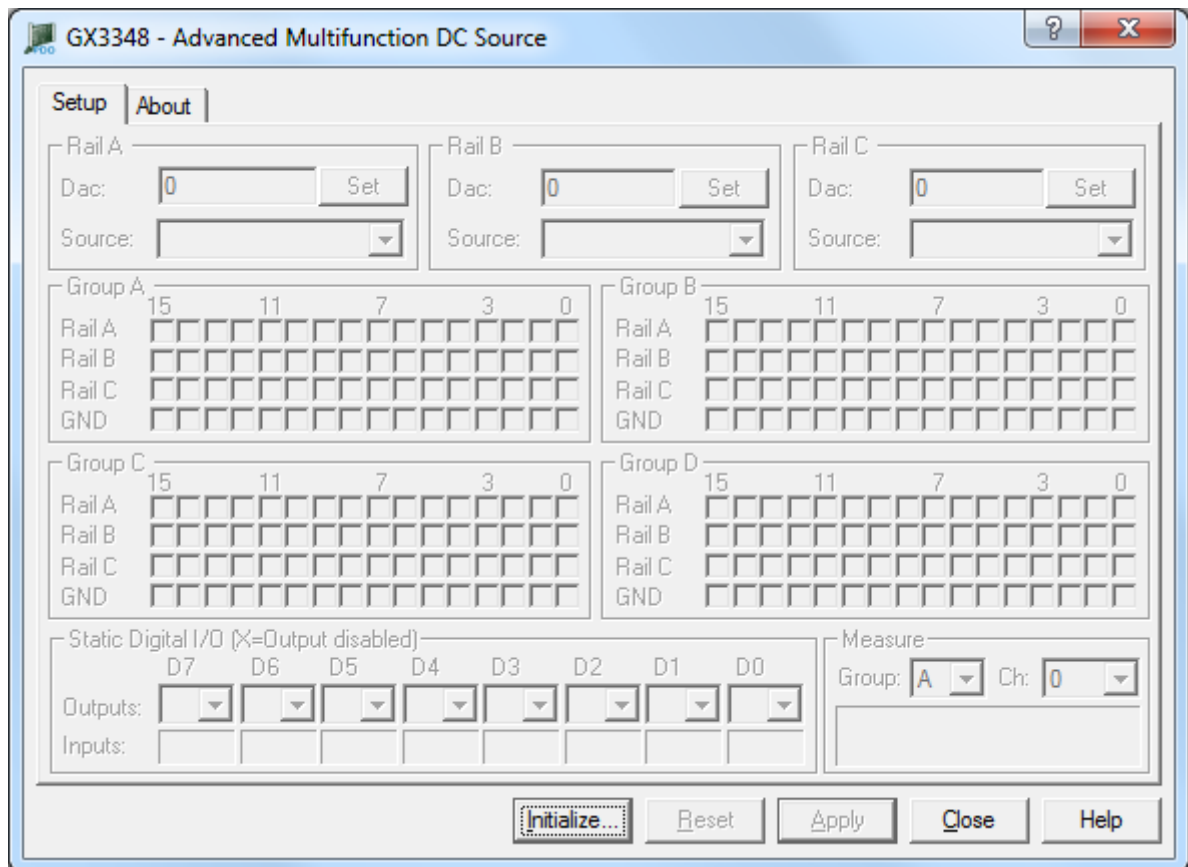
In order to calibrate the GX3348 the user must have access to a 6 ½ digit reference **DMM** (for VDC measurements).

### Calibration Licensing

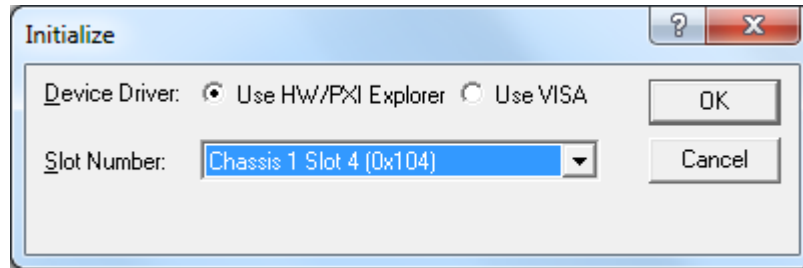
A calibration license may be obtained from Marvin Test Solutions in order to unlock the Calibration functionality.

The software front panel used to enter a valid license string using the following procedure:

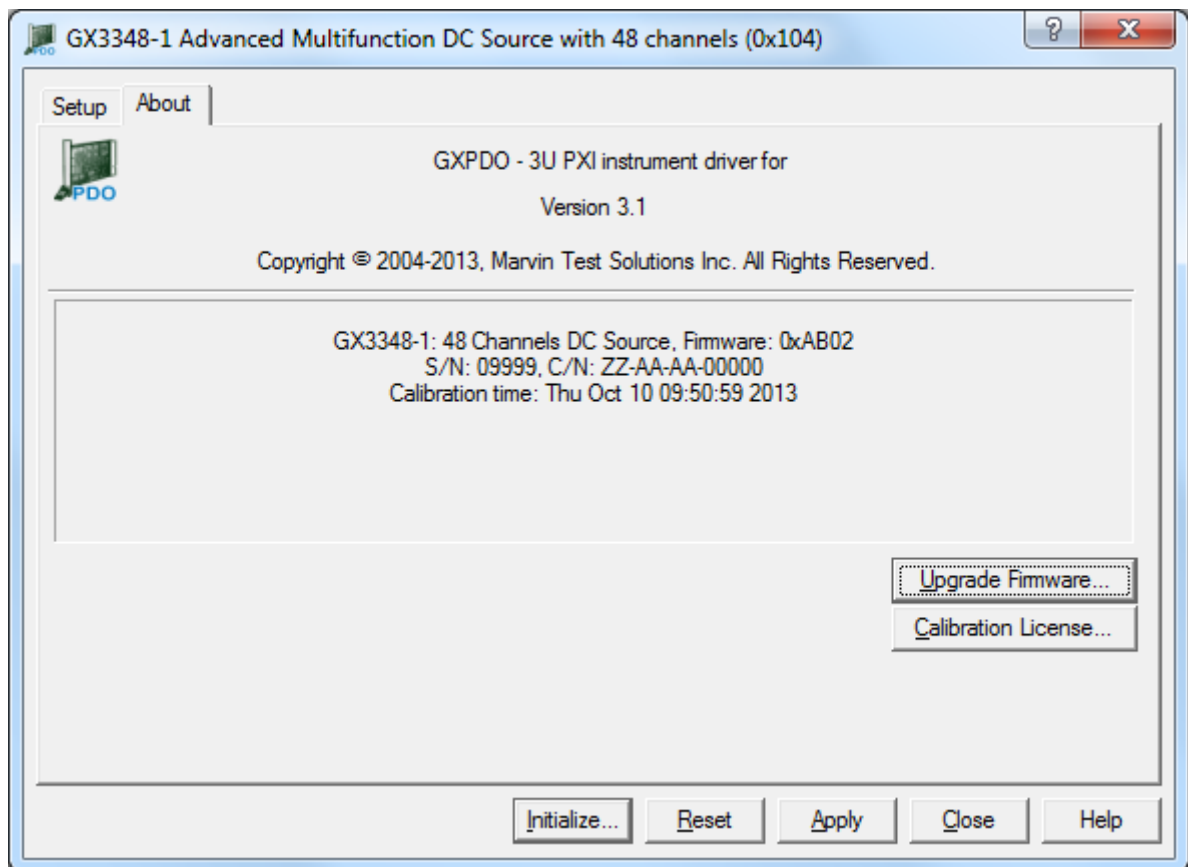
1. Initialize the software front panel:



**Figure 5-1: Initialize the Software**



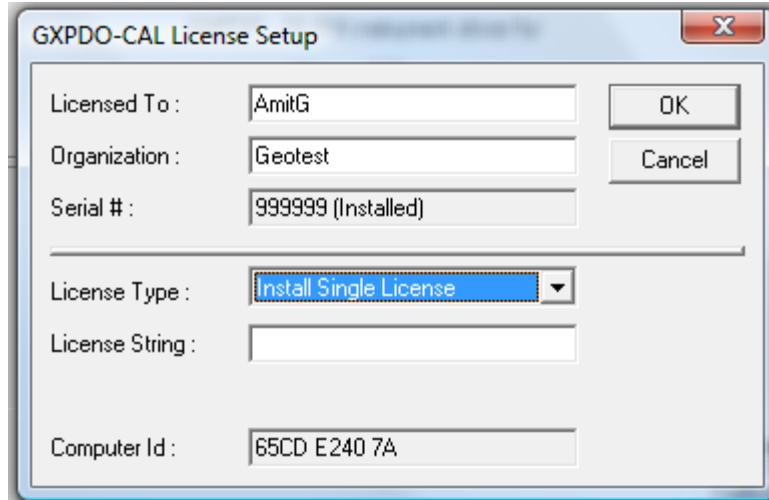
2. Click on the About Tab:



**Figure 5-2: Calibration Tab**

3. Click on the Calibration License... button.
4. Make note of the Computer ID. Send this ID to Marvin Test Solutions in exchange for a License String.
5. Fill in the appropriate fields including the License String and click OK.





**Figure 5-3: License Setup**

At this point, the Calibration functionality has been unlocked on a specific computer.

### **GXPDO Functions used for Calibration**

---

The functions used for calibration are:

<b>Function</b>	<b>Description</b>
<b>Gx3348Initialize</b>	Initializes the GX3348 Pass the hex value 0x1234 to the <i>pnStatus</i> parameter to unlock calibration functionality.
<b>Gx3348CalAdc</b>	Calibrate ADC Channel
<b>Gx3348CalDac</b>	Calibrate DAC
<b>Gx3348CalSetDacVoltagePoint</b>	Set Voltage Point when calibrating DACs
<b>Gx3348CalSetMode</b>	Sets and enables calibration mode prior to calibration procedure
<b>Gx3348CalWriteEEPROM</b>	Finalizes the calibration by writing the DAC and Measure (ADC) Offsets and Gains to EEPROM

**Table 5-1: GXPDO Calibration Functions**

For further information on these functions, review the **Function Reference** section of this manual.

## Calibration Procedure

---

Use the CalEasy user guide for connection information to perform this procedure.

### Initial Calibration Procedure

1. Initialize the GX3348 (**Gx3348Initialize**)
2. Reset GX3348
3. Connect Reference DMM to Group A Channel 0 of the GX3348

### DAC Voltage Calibration

1. Set DAC Calibration Voltage point to Positive by calling **Gx3348CalSetDacVoltagePoint**
2. Use reference DMM to measure voltage
3. Set DAC Calibration Voltage point to Zero by calling **Gx3348CalSetDacVoltagePoint**
4. Use reference DMM to measure voltage
5. Set DAC Calibration Voltage point to Negative by calling **Gx3348CalSetDacVoltagePoint**
6. Use reference DMM to measure voltage
7. Pass measurements from step 2, 4, and 6 to driver by calling **Gc3348CalDac**
8. Repeat steps 1 to 7 for Rails A, B, and C.

### ADC Calibration

1. Connect Reference DMM to Gx3348 channel that is to be calibrated
2. Close relay to connect Rail A to the channel to be calibrated by calling **Gx3348SetRelay**
3. Set Calibration Voltage point to Positive by calling **Gx3348CalSetDacVoltagePoint**
4. Use reference DMM to measure voltage
5. Measure ADC Voltage by calling **Gx3348Measure**
6. Set Calibration Voltage point to Negative by calling **Gx3348CalSetDacVoltagePoint**
7. Use reference DMM to measure voltage
8. Measure ADC Voltage by calling **Gx3348Measure**
9. Pass measurements from step 4, 5, 7 and 8 to driver by calling **Gx3348CalDac**
10. Repeat steps 1 to 9 for channels 0-15 on Groups A-D on the instrument

### Final Calibration Procedure

1. Write the User Calibration Set to EEPROM using the **Gx3348CalWriteEEPROM** function.

## Chapter 6 - Functions Reference

### Introduction

---

The functions reference chapter organizes the list of GX3348 driver functions in an alphabetical order. Each function description contains the function name; purpose, syntax, parameters description and type followed by Comments, an Example (written in C), and a See Also sections.

All function and parameter syntax follow the same rules:

- Strings are ASCIIZ (null or zero character terminated).
- The first parameter of most functions is *nHandle* (16-bit integer). This parameter is required for operating the board and is returned by the **Gx3348Initialize** function. The *nHandle* is used to identify the board when calling a function for programming and controlling the operation of that board.
- All functions return a status with the last parameter named *pnStatus*. The *pnStatus* is zero if the function was successful, or non-zero on error. The description of the error is available using the **GxPdoGetErrorString** function or by using a predefined constant, defined in the driver interface files: GXPDO.H, GXPDO.BAS, GXPDO.VB, GXPDO.PAS or GX3348.DRV.
- Parameter name are prefixed as follows:

Prefix	Type	Example
<i>a</i>	Array - prefix this before the simple type.	<i>anArray</i> (Array of Short)
<i>b</i>	BOOL – Boolean, 0 for FALSE; <>0 for TRUE	<i>bUpdate</i>
<i>d</i>	DOUBLE - 8 bytes floating point	<i>dReading</i>
<i>dw</i>	DWORD - double word (unsigned 32-bit)	<i>dwTimeout</i>
<i>hwnd</i>	Window handle (32-bit integer).	<i>hwndPanel</i>
<i>l</i>	LONG - (signed 32-bit)	<i>lBits</i>
<i>n</i>	SHORT - (signed 16-bit)	<i>nMode</i>
<i>p</i>	Pointer - Usually used to return a value. Prefix this before the simple type.	<i>pnStatus</i>
<i>sz</i>	Null - (zero value character) terminated string	<i>szMsg</i>
<i>uc</i>	BYTE - (8 bits) unsigned.	<i>ucValue</i>
<i>w</i>	WORD - Unsigned short (unsigned 16-bit)	<i>wParam</i>

**Table 6-1: Parameter Name Prefixes**

## GX3348 Functions

The following list is a summary of functions available for GX3348:

Driver Functions	Description
<b>General</b>	
Gx3348Initialize	Initializes the driver for the specified slot using the HW device driver.
Gx3348InitializeVisa	Initializes the driver for the specified slot using VISA.
Gx3348Panel	Opens a virtual panel used to interactively control the GX3348.
Gx3348Reset	Resets the GX3348 board to its default settings.
GxPdoGetDriverSummary	Returns the driver name and version.
GxPdoGetErrorString	Returns the error string associated with the specified error number.
<b>Functions</b>	
Gx3348GetChannelRail	Returns the specified group's channel rail connection.
Gx3348GetDac	Returns the specified rail's DAC voltage.
Gx3348GetDigitalInputs	Reads back the eight digital inputs lines logical states.
Gx3348GetDigitalOutputs	Returns back the eight digital output lines output enables and values.
Gx3348GetRailSource	Returns the specified rail source
Gx3348Measure	Returns the voltage measurement at a channel
Gx3348SelfTest	Performs a self-test on the board Relays and DACs
Gx3348SetChannelRail	Sets the specified group's channel rail connection.
Gx3348SetDac	Returns the specified rail's DAC voltage.
Gx3348SetDigitalOutputs	Sets the eight digital output lines output enables and values.
Gx3348SetRailSource	Set the specified rail source.
<b>Calibration</b>	
Gx3348CalAdc	Calibrates each channel's ADC
Gx3348CalDac	Calibrates each DAC
Gx3348CalSetDacVoltagePoint	Sets the calibration voltage point for the DAC
Gx3348CalSetMode	Sets the calibration mode
Gx3348CalWriteEEPROM	Finalizes calibration and writes DAC and ADC gains and offsets to the EEPROM.

## Gx3348CalAdc

---

### Purpose

Calibrates an ADC Channel

### Syntax

**Gx3348CalAdc** (*nHandle*, *nGroup*, *nChannel*, *dRefPositive*, *dRefNegative*, *dMeasurePositive*, *dMeasureNegative*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nGroup</i>	SHORT	Selects the group to calibrate <ol style="list-style-type: none"> <li>0. GX3348_GROUP_A: Calibrates an ADC Channel from Group A</li> <li>1. GX3348_GROUP_B: Calibrates an ADC Channel from Group B</li> <li>2. GX3348_GROUP_C: Calibrates an ADC Channel from Group C</li> <li>3. GX3348_GROUP_D: Calibrates an ADC Channel from Group D</li> </ol>
<i>nChannel</i>	DOUBLE	Selects the ADC channel within a group to calibrate
<i>dRefPositive</i>	DOUBLE	Sets the positive measurement taken from an external DMM
<i>dRefNegative</i>	DOUBLE	Sets the negative measurement taken from an external DMM
<i>dMeasurePositive</i>	DOUBLE	Sets the positive measurement taken from the ADC channel
<i>dMeasureNegative</i>	DOUBLE	Sets the negative measurement taken from the ADC channel
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function calibrates and ADC channel by using the internal DAC to source voltages. Calibration of the ADC does require external DMM. Calling **Gx3348CalWriteEEPROM** will finalize the calibration.

**Example**

The following example calibrates all ADC channels:

```

SHORT nHandle, nStatus, nGroup, nChannel;
DOUBLE dMeasurePositive, dMeasureNegative, dRefPositive, dRefNegative;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348CalSetMode(nHandle, GX3348_CAL_MODE_ENABLED, &nStatus);
for(nGroup=0; nGroup<3; nGroup++)
{
    for(nChannel=0; nChannel<16; nChannel++)
    {

Gx3348SetRailSource(nHandle, GX3348_RAIL_A, GX3348_RAIL_SOURCE_INTERNAL_DAC, &nStatus);
Gx3348SetChannelRail(nHandle, nGroup, nChannel, GX3348_RAIL_A, &nStatus)
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_POSITIVE,
&nStatus);
DMMMeasure(&dRefPositive);
Gx3348Measure(nHandle, nGroup, nChannel, &dMeasurePositive, &nStatus);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_NEGATIVE,
&nStatus);
DMMMeasure(&dMeasureNegative);
Gx3348Measure(nHandle, nGroup, nChannel, &dMeasureNegative, &nStatus);
Gx3348CalAdc(nHandle, nGroup, nChannel, dRefPositive, dRefNegative, dMeasurePositive,
dMeasureNegative, &nStatus);
    }
}
Gx3348CalWriteEEPROM(nHandle, &nStatus);

```

**See Also**

**Gx3348CalDac, Gx3348CalSetDacVoltagePoint, Gx3348CalWriteEEPROM, GxPdoGetErrorString**

## Gx3348CalDac

---

### Purpose

Calibrates a DAC

### Syntax

**Gx3348CalDac** (*nHandle*, *nDac*, *dRefPositive*, *dRefNegative*, *dRefZero*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nDac</i>	SHORT	Set the DAC that will be configured 0. GX3348_RAIL_DAC_A: Calibrate DAC A 1. GX3348_RAIL_DAC_B: Calibrate DAC B 2. GX3348_RAIL_DAC_C: Calibrate DAC C
<i>dRefPositive</i>	DOUBLE	Sets the positive measurement taken from an external DMM
<i>dRefNegative</i>	DOUBLE	Sets the negative measurement taken from an external DMM
<i>dRefZero</i>	DOUBLE	Sets the zero measurement taken from an external DMM
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function calibrates a DAC by passing in three measurements taken using a reference DMM at three calibration points (positive, zero and negative). A Voltage point is selected by calling **Gx3348CalSetDacVoltagePoint**. Calling **Gx3348CalWriteEEPROM** will finalize the calibration.

### Example

The following example calibrates DAC A:

```
SHORT nHandle, nStatus;
DOUBLE dRefPositive, dRefNegative, dRefZero;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348CalSetMode(nHandle, GX3348_CAL_MODE_ENABLED, &nStatus);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_POSITIVE,
&nStatus);
DMMMeasure(&dRefPositive);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_ZERO,
&nStatus);
DMMMeasure(&dRefZero);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_NEGATIVE,
&nStatus);
DMMMeasure(&dRefNegative);

Gx3348CalDac(nHandle, GX3348_RAIL_DAC_A, dRefPositive, dRefNegative, dRefZero, &nStatus);
Gx3348CalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**Gx3348CalSetDacVoltagePoint**, **Gx3348CalWriteEEPROM**, **GxPdoGetErrorString**

## Gx3348CalSetDacVoltagePoint

---

### Purpose

Sets a DAC's offset and gain before writing it to the EEPROM

### Syntax

**Gx3348CalSetDacVoltagePoint** (*nHandle*, *nDac*, *nCalVoltagePoint*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nDac</i>	SHORT	Select the DAC to calibrate 0. GX3348_RAIL_DAC_A: Calibrate DAC A 1. GX3348_RAIL_DAC_B: Calibrate DAC B 2. GX3348_RAIL_DAC_C: Calibrate DAC C
<i>nCalVoltagePoint</i>	SHORT	Sets the positive measurement taken from an external DMM 0. GX3348_CAL_DAC_VOLTAGE_POINT_ZERO: Calibrate DAC A 1. GX3348_CAL_DAC_VOLTAGE_POINT_NEGATIVE: Calibrate DAC B 2. GX3348_CAL_DAC_VOLTAGE_POINT_POSITIVE: Calibrate DAC C
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function sets a voltage point for DAC Calibration. Channel 0 of Group A should be measured by a reference DMM at each of the three voltage points (Zero, Negative, and Positive voltages). These measurements will be passed into the **Gx3348CalDac** function. Calling **Gx3348CalWriteEEPROM** will finalize the calibration.

### Example

The following example calibrates DAC A:

```
SHORT nHandle, nStatus;
DOUBLE dRefPositive, dRefNegative, dRefZero;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348CalSetMode(nHandle, GX3348_CAL_MODE_ENABLED, &nStatus);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_POSITIVE,
&nStatus);
DMMMeasure (&dRefPositive);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_ZERO,
&nStatus);
DMMMeasure (&dRefZero);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_NEGATIVE,
&nStatus);
DMMMeasure (&dRefNegative);

Gx3348CalDac(nHandle, GX3348_RAIL_DAC_A, dRefPositive, dRefNegative, dRefZero, &nStatus);
Gx3348CalWriteEEPROM(nHandle, &nStatus);
```



**See Also**

**Gx3348CalSetDacVoltagePoint, Gx3348CalWriteEEPROM, GxPdoGetErrorString**

## Gx3348CalSetMode

---

### Purpose

Sets the calibration constants to a default value before starting the calibration procedure

### Syntax

**Gx3348CalSetMode** (*nHandle*, *nCalibrationMode*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nCalibrationMode</i>	SHORT	Enables or disables the calibration mode: 0. GX3348_CAL_MODE_DISABLED: Disables calibration mode 1. GX3348_CAL_MODE_ENABLED: Enables calibration mode
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function sets the driver to calibration mode. Calibration mode causes the ADC to use default gain and offset constants.

### Example

The following example calibrates DAC A:

```
SHORT nHandle, nStatus;
DOUBLE dRefPositive, dRefNegative, dRefZero;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348CalSetMode(nHandle, GX3348_CAL_MODE_ENABLED, &nStatus);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_POSITIVE,
&nStatus);
DMMMeasure(&dRefPositive);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_ZERO,
&nStatus);
DMMMeasure(&dRefZero);
Gx3348CalSetDacVoltagePoint(nHandle, GX3348_RAIL_DAC_A, GX3348_CAL_DAC_VOLTAGE_POINT_NEGATIVE,
&nStatus);
DMMMeasure(&dRefNegative);

Gx3348CalDac(nHandle, GX3348_RAIL_DAC_A, dRefPositive, dRefNegative, dRefZero, &nStatus);
Gx3348CalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**Gx3348CalSetDacVoltagePoint**, **Gx3348CalWriteEEPROM**, **GxPdoGetErrorString**

## Gx3348CalWriteEEPROM

---

### Purpose

Finalizes the calibration by writing the DAC and Measure (ADC) Offsets and Gains to EEPROM

### Syntax

**Gx3348CalWriteEEPROM** (*nHandle*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function writes all DACs gain and offset values and the new calibration date and time to the on-board EEPROM.

Calling this function completes the calibration process.

### Example

The following example initializes and writes the calibration data to the EEPROM to finalize the calibration:

```
SHORT nHandle, nStatus;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348CalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**Gx3348CalDac**, **Gx3348CalAdc**, **GxPdoGetErrorString**

## Gx3348GetBoardSummary

---

### Purpose

Returns the board information.

### Syntax

**Gx3348GetBoardSummary**(*nHandle*, *pszSummary*, *nSumMaxLen*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>pszSummary</i>	PSTR	Buffer to contain the returned board info (null terminated) string.
<i>nSumMaxLen</i>	SHORT	Size of the buffer to contain the string.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The dc supply summary string provides the following data from the board in the order shown:

- Instrument Name (e.g. Gx3348)
- Firmware Revision (e.g. 1.00)
- Serial Number (e.g. '00012')
- Control Number (e.g. 'CA-CA-000')

For example, the returned string looks like the following:

GX3348: DC Source, Firmware: 0xC, S/N: 00012, C/N CA-CA-000"

### Example

The following example returns the board summary:

```
SHORT nHandle, nStatus;
CHAR szSummary [256];

Gx3348GetBoardSummary(nHandle, szSummary, sizeof(szSummary), &nStatus);
printf("The Board Summary is %s", szSummary);
```

### See Also

**GxPdoGetDriverSummary**, **Gx3348Initialize**, **GxPdoGetErrorString**

## Gx3348GetCalibrationInfo

---

### Purpose

Returns the calibration information.

### Syntax

**Gx3348GetCalibrationInfo** (*nHandle*, *pszCalibrationInfo*, *nInfoMaxLen*, *pnDaysUntilExpire*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>pszCalibrationInfo</i>	PSTR	Buffer to contain the returned board's calibration information (null terminated) string.
<i>nInfoMaxLen</i>	SHORT	Size of the buffer to contain the error string.
<i>pnDaysUntilExpire</i>	PSHORT	Returns the number of days until or from expiration, if number is > 0 then calibration is current otherwise past due.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The returned board's calibration information has the following fields:

Model: model number, e.g. "GX3348"

Serial Number: serial number, e.g. 216

Control Number: Marvin Test Solutions control number, e.g. "\*-CH-CB-0"

Production Calibration Date: Wed Oct 24 12:30:25 2010

Calibration Date: Wed Oct 24 12:31:58 2010

Recommended Interval: 1 year

Next Calibration Date: Fri Oct 24 12:31:58 2011

Status: calibration status can be either "Expired" followed by the number of days past expiration or "Current" followed by number of days until expire.

Calibration License: can be either "Installed" with the calibration license number or "Not Installed".

## Example

The following example returns the board's calibration information string:

```
SHORT  nStatus;
char    szCalibrationInfo[1024];
BOOL    bExpired;

Gx3348GetCalibrationInfo(nHandle, szCalibrationInfo, sizeof szCalibrationInfo, &bExpired,
&nStatus);

szCalibrationInfo string printout:

Model: GX3348
Serial Number: 12
Control Number: *-CA-CA-00
Production Calibration Date: Wed May 23 12:30:25 2012
Calibration Date: Wed Oct 24 12:31:58 2007
Recommended Interval: 1 year
Next Calibration Date: Fri May 24 12:31:58 2013
Status: Expired (891 days past expiration)
Calibration License: Installed license number 3
```

## See Also

**Gx3348Initialize, Gx3348GetBoardSummary, GxPdoGetErrorString**

## Gx3348GetChannelRail

---

### Purpose

Returns the specified group's channel rail connection.

### Syntax

**Gx3348GetChannelRail** (*nHandle*, *nGroup*, *nChannel*, *pnRail*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>nGroup</i>	SHORT	Channel Group to measure 0. GX3348_GROUP_A 1. GX3348_GROUP_B 2. GX3348_GROUP_C 3. GX3348_GROUP_D
<i>nChannel</i>	SHORT	Specified group's channel number 0-15.
<i>pnRail</i>	PSHORT	Returned Rail connections are as follow: 0. GX3348_RAIL_A: connect to rail A 1. GX3348_RAIL_B: connect to rail B 2. GX3348_RAIL_C: connect to rail C 3. GX3348_GROUND: : connect to ground -1 GX3348_RAIL_NONE: no rail is connected.
<i>PnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The Gx3348 has two models: Gx3348-1 with 48 channels or Gx3348-1 with 64 channels. The channels are divided into 4 groups, with each group consisting of 16 channels. Each channel can be connected to one of three DAC rails or to a Ground rail or disconnected from all rails (GX3348\_RAIL\_NONE).

### Example

The following example returns the rail connection for group A channel 2:

```
SHORT nRail, nStatus;
Gx3348GetChannelRail(nHandle, GX3348_GROUP_A, 2, &nRail, &nStatus);
```

### See Also

**Gx3348SetChannelRail**, **Gx3348SetDac**, **Gx3348SetDigitalOutputs**, **Gx3348SetRailSource**, **GxPdoGetErrorString**

## Gx3348GetDac

---

### Purpose

Returns the specified DAC's voltage.

### Syntax

**Gx3348GetDac** (*nHandle*, *nDac*, *pdVoltage*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nDac</i>	SHORT	DAC to set. 0. GX3348_DAC_RAIL_A: dedicated Rail A DAC 1. GX3348_DAC_RAIL_B: dedicated Rail B DAC. 2. GX3348_DAC_RAIL_C: dedicated Rail C DAC.
<i>pdVoltage</i>	PDOUBLE	Returns DAC Voltage (-20 to 30V are valid)
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Each DAC can be programmed with a voltage ranging between -20V to 30V.

### Example

The following example initializes the board, returns the rail connecting Channel 12 of Group A, and returns DAC B voltage:

```
SHORT nRail, nStatus;

DOUBLE dVoltage;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348GetChannelRail(nHandle, GX3348_GROUP_A, 12, &nRail, &nStatus);

Gx3348GetDac(nHandle, GX3348_DAC_RAIL_B, &dVoltage, &nStatus);
if(nRail== GX3348_RAIL_B)
    printf("Channel 12 in Group A is connected to DAC B in Rail B");
See Gx3348SetRailSource API for a comprehensive C example.
```

### See Also

**Gx3348SetChannelRail**, **Gx3348SetDac**, **Gx3348SetDigitalOutputs**, **Gx3348SetRailSource**, **GxPdoGetErrorString**



## Gx3348GetRailSource

---

### Purpose

Returns the specified rail's source.

### Syntax

**Gx3348GetRailSource** (*nHandle*, *nRail*, *pnRailSource*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>nRail</i>	SHORT	Rail number: 0. GX3348_RAIL_A: connect to rail A 1. GX3348_RAIL_B: connect to rail B 2. GX3348_RAIL_C: connect to rail C 3. GX3348_GROUND: connect to ground -1 GX3348_RAIL_NONE: no rail is connected.
<i>pnRailSource</i>	PSHORT	Rail source: 0. GX3348_RAIL_SOURCE_INTERNAL_DAC: Internal rail dac. All three on-board rails (A through C), have a dedicated DAC that can be set from -20V to +32V. 1. GX3348_RAIL_SOURCE_EXTERNAL rails A and C can be connected to an external source via the front J6 68 pin connector, 2. GX3348_RAIL_SOURCE_INTERNAL_5V: Rail B can be connected to an on-board 5V source. 3. GX3348_RAIL_SOURCE_EXTERNAL_AMPLIFIED: Rail A can be connected to an external source which will be amplified by a nominal gain of 6.6 and can amplify signals from DC to > 10 KHz.
<i>PnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Rails sources are as follow:

GX3348\_RAIL\_A:

0. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail A dac can be set from -20V to +32V.
1. GX3348\_RAIL\_SOURCE\_EXTERNAL: rail A can be connected to an external source via the front J6 68 pin connector,
3. GX3348\_RAIL\_SOURCE\_EXTERNAL\_AMPLIFIED: rail A can be connected to an external source which will be amplified by a factor of 3.

GX3348\_RAIL\_B:

0. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail B dac can be set from -20V to +32V.
2. GX3348\_RAIL\_SOURCE\_INTERNAL\_5V: Rail B can be connected to an on-board 5V source.

#### GX3348\_RAIL\_C:

0. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail C dac can be set from -20V to +32V.
1. GX3348\_RAIL\_SOURCE\_EXTERNAL rail C can be connected to an external source via the front J6 68 pin connector.

#### Example

The following example returns rail A source:

```
SHORT nRailsource, nStatus;
```

```
Gx3348GetRailSource (nHandle, GX3348_RAIL_A, &nRailsource, &nStatus);  
See Gx3348SetRailSource API for a comprehensive C example.
```

#### See Also

**Gx3348SetChannelRail, Gx3348SetDac, Gx3348SetDigitalOutputs, Gx3348SetRailSource, GxPdoGetErrorString**

## Gx3348GetDigitalInputs

---

### Purpose

Reads back the eight digital inputs lines states.

### Syntax

**Gx3348GetDigitalInputs** (*nHandle*, *pdwData*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>pdwData</i>	PDWORD	Bits 0-7 represent the 8 digital inputs. Bit zero corresponds to digital I/O line 0.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Each of the 8 Output digital lines is also connected to an input line. If the output line is not enabled, then an external digital input can be monitored using this function. Otherwise if the line output is enabled, the function returns the output line logical state.

### Example

The following example sets the lower 4 digital outputs lines to hi and the upper 4 lines to low, all odd lines (1, 3, etc) outputs are then enabled. The digital lines actual states are then read back.

```
SHORT nStatus;
DWORD dwOutputsEnabled, dwData;
DWORD dwDigitalInputs;

Gx3348SetDigitalOutputs (nHandle, 0x55, 0x0F, &nStatus);
Gx3348GetDigitalOutputs (nHandle, &dwOutputsEnabled, &dwData, &nStatus);
```

### See Also

**Gx3348SetDigitalOutputs**, **Gx3348GetDigitalOutputs**, **GxPdoGetErrorString**

## Gx3348GetDigitalOutputs

---

### Purpose

Reads back the eight digital output lines output enables and values.

### Syntax

**Gx3348GetDigitalOutputs** (*nHandle*, *pdwOutputsEnabled*, *pdwData*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>pdwOutputsEnabled</i>	PDWORD	Bits 0-7 represent the 8 digital outputs lines inputs. Bit zero corresponds to digital I/O line 0. Bit high will enable the specified digital output. Bit low will disable the specified digital output (default after reset).
<i>pdwData</i>	PDWORD	Bits 0-7 represent the 8 digital inputs. Bit zero corresponds to digital I/O line 0. Bit high will output logic high. Bit low will output a logic low (default after reset).
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Each of the 8 Output digital lines is also connected to an input line. If the output line is not enabled, then an external digital input can be monitored using this function. Otherwise if the line output is enabled the function returns the output line logical state.

### Example

The following example sets the lower 4 digital outputs lines to hi and the upper 4 lines to low, all odd lines (1, 3, etc) outputs are then enabled. The digital lines actual states are then read back.

```
SHORT nStatus;
DWORD dwOutputsEnabled, dwData;
DWORD dwDigitalInputs;

Gx3348SetDigitalOutputs (nHandle, 0x55, 0x0F, &nStatus);
Gx3348GetDigitalOutputs (nHandle, &dwOutputsEnabled, &dwData, &nStatus);
Gx3348GetDigitalInputs (nHandle, &dwDigitalInputs, &nStatus);
```

### See Also

**Gx3348SetDigitalOutputs**, **Gx3348GetDigitalInputs**, **GxPdoGetErrorString**

## Gx3348Initialize

---

### Purpose

Initializes the driver for the specified PXI slot using the HW device driver.

### Syntax

**Gx3348Initialize** (*nSlot*, *pnHandle*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nSlot</i>	SHORT	GX3348 board slot number. See Comments.
<i>pnHandle</i>	PSHORT	Returned Handle for the PXI board.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, 1 on failure.

### Comments

The Marvin Test Solutions HW device driver is installed with the driver and is the default device driver. The function returns a handle that for use with other functions to program the board. The function does not change any of the board settings.

The specified PXI slot number is displayed by the **PXI/PCI Explorer** applet that can be opened from the Windows **Control Panel**. You may also use the label on the chassis below the PXI slot where the board is installed. The function accepts two types of slot numbers:

- A combination of chassis number (chassis # x 256) with the chassis slot number. For example, 0x105 (chassis 1 slot 5).
- Legacy nSlot as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet (1-255).

### Example

The following example initializes a board at PXI chassis 2 slot 7.

```
SHORT nHandle, nStatus;

Gx3348Initialize(0x207, &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

### See Also

**Gx3348InitializeVisa**, **GxPdoGetErrorString**, **GxPdoReset**

## Gx3348InitializeVisa

---

### Purpose

Initializes the driver for the specified PXI slot using the default VISA provider.

### Syntax

**Gx3348InitializeVisa** (*szVisaResource*, *pnHandle*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>szVisaResource</i>	PCSTR	String identifying the location of the specified board in order to establish a session.
<i>pnHandle</i>	PSHORT	Returned Handle (session identifier) that can be used to call any other operations of that resource
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, 1 on failure.

### Comments

The **Gx3348InitializeVisa** opens a VISA session to the specified resource. The function uses the default VISA provider configured in your system to access the board. You must ensure that the default VISA provider support PXI/PCI devices and that the board is visible in the VISA resource manager before calling this function.

The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI\_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as follows:

- Using chassis, slot: "PXI0::CHASSIS1::SLOT5"
- Using the PCI Bus/Device combination: "PXI9::13::INSTR" (bus 9, device 9).
- Using alias: "PDO1". Use the PXI/PCI Explorer to set the device alias.

The function returns a board handle (session identifier) that can be used to call any other operations of that resource. The session is opened with `VI_TMO_IMMEDIATE` and `VI_NO_LOCK` VISA attributes. On terminating the application the driver automatically invokes `viClose()` terminating the session.

### Example

The following example initializes a DC Power Supply board at PXI bus 5 and device 11.

```
SHORT nHandle, nStatus;

Gx3348InitializeVisa("PXI5::11::INSTR", &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

### See Also

**Gx3348Initialize**, **GxPdoGetErrorString**, **Gx3348Reset**

## Gx3348Measure

---

### Purpose

Measure voltage at channel pin

### Syntax

**Gx3348Measure** (*nHandle*, *nGroup*, *nChannel*, *pdMeasurement*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nGroup</i>	SHORT	Channel Group to measure 4. GX3348_GROUP_A: Connect/Disconnect a channel from Group A 5. GX3348_GROUP_B: Connect/Disconnect a channel from Group B 6. GX3348_GROUP_C: Connect/Disconnect a channel from Group C 7. GX3348_GROUP_D: Connect/Disconnect a channel form Group D
<i>nChannel</i>	SHORT	Sets Channel that will be measured (0-15)
<i>pdMeasurement</i>	PDOUBLE	Returns the voltage measurement of a channel.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

This function measures the voltage present at each of the up to 64 output channels. The hardware uses an ADC to measure voltage

### Example

The following example initializes and measures the voltage present at Channel 10 of Group B:

```
SHORT nHandle, nStatus;
DOUBLE dMesaurement;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348Measure (nHandle, GX3348_RAIL_DAC_B, 10, &dMeasurement, &nStatus);
```

### See Also

**Gx3348Initialize**, **Gx3348SetDac**, **Gx3348GetDac**, **GxPdoGetErrorString**

## Gx3348Panel

---

### Purpose

Opens a virtual panel used to interactively control the power source.

### Syntax

**Gx3348Panel** (*pnHandle*, *hwndParent*, *nMode*, *phwndPanel*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>pnHandle</i>	PSHORT	Handle for the PXI board.
<i>hwndParent</i>	HWND	Panel parent window handle. A value of 0 sets the desktop as the parent window.
<i>nMode</i>	SHORT	The mode in which the panel main window is created. 0 for modeless window and 1 for modal window.
<i>phwndPanel</i>	HWND	Returned window handle for the panel.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The function is used to create the panel window. The panel window may be open as a modal or a modeless window depending on the *nMode* parameters.

If the mode is set to modal dialog (*nMode*=1), the panel will disable the parent window (*hwndParent*) and the function will return only after the window closes. In that case, the *pnHandle* may return the handle created by the user using the panel Initialize dialog. This handle may be used when calling other board functions.

If a modeless dialog was created (*nMode*=0), the function returns immediately after creating the panel window returning the handle to the panel - *phwndPanel*. It is the responsibility of calling program to dispatch windows messages to this window so that the window can respond to messages.

### Example

The following example opens the panel in modal mode:

```
DWORD dwPanel;
SHORT nHandle=0, nStatus;

GxPdoPanel(&nHandle, 0, 1, &dwPanel, &nStatus);
```

### See Also

**Gx3348Initialize**, **GxPdoGetErrorString**



## Gx3348Reset

---

### Purpose

Resets the power source.

### Syntax

**Gx3348Reset** (*nHandle*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Default settings are as follow:

- All Channel Relays: Open
- External DAC: Disconnected
- DAC Voltages: 0.0 V

### Example

The following example initializes and resets the board:

```
SHORT nHandle, nStatus;  
  
Gx3348Initialize (1, &nHandle, &nStatus);  
Gx3348Reset (nHandle, &nStatus);
```

### See Also

**Gx3348Initialize**, **GxPdoGetErrorString**

## Gx3348SelfTest

---

### Purpose

Runs a self-test on the board

### Syntax

**Gx3348SelfTest** (*nHandle*, *pbSuccess*, *pszTestResult*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>pbSuccess</i>	PBOOL	Returns the resulting pass/fail result
<i>pszTestResult</i>	PSTR	Returns a detailed message regarding the self-test result
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The self-test checks each relay for voltage generated from each of the 3 DACs.

The function returns a string, *pszTestResult*, that contains an explanation of any error that occurred. For example, if the voltage measurement fails at a certain channel, *pszTestResult* could return the following: "Selftest Failed: Relay failed at Rail A, Group B, Channel 4"

### Example

The following example performs a self-test to the board:

```
SHORT nHandle, nStatus;

BOOL    bSuccess;
CHAR    szTestResult[512];

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348SelfTest (nHandle, &bSuccess, szTestResult, &nStatus);
printf("The test result is: %s", szTestResult);
```

### See Also

**Gx3348Initialize**, **GxPdoGetErrorString**

## Gx3348SetChannelRail

---

### Purpose

Sets the specified group's channel rail connection.

### Syntax

**Gx1838SetChannelRail** (*nHandle*, *nGroup*, *nChannel*, *nRail*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>nGroup</i>	SHORT	Channel Group to measure 0. GX3348_GROUP_A 1. GX3348_GROUP_B 2. GX3348_GROUP_C 3. GX3348_GROUP_D
<i>nChannel</i>	SHORT	Specified group's channel number 0-15.
<i>nRail</i>	SHORT	Rail connections are as follow: 0. GX3348_RAIL_A: connect to rail A 1. GX3348_RAIL_B: connect to rail B 2. GX3348_RAIL_C: connect to rail C 3. GX3348_GROUND: connect to ground -1 GX3348_RAIL_NONE: no rail is connected.
<i>PnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The Gx3348 has two models: Gx3348-1 with 48 channels or Gx3348-1 with 64 channels. The channels are divided into 4 groups, with each group consisting of 16 channels. Each channel can be connected to one of three DAC rails or to a Ground rail or disconnected from all rails (GX3348\_RAIL\_NONE).

### Example

The following example returns the rail connection for group A channel 2:

```
SHORT nRail, nStatus;
```

```
Gx3348GetChannelRail(nHandle, GX3348_GROUP_A, 2, &nRail, &nStatus);  
See Gx3348SetRailSource API for a comprehensive C example.
```

### See Also

**Gx3348GetChannelRail**, **Gx3348SetDac**, **Gx3348SetDigitalOutputs**, **Gx3348SetRailSource**, **GxPdoGetErrorString**

## Gx3348SetDac

---

### Purpose

Sets the specified DAC's voltage

### Syntax

**Gx3348SetDac** (*nHandle*, *nDac*, *dVoltage*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle for the PXI board.
<i>nRail</i>	SHORT	DAC to set. 0. GX3348_DAC_RAIL_A: dedicated Rail A DAC. 1. GX3348_DAC_RAIL_B: dedicated Rail B DAC. 2. GX3348_DAC_RAIL_C: dedicated Rail C DAC.
<i>dVoltage</i>	DOUBLE	DAC Voltage (-20 to 30V are valid)
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Each DAC can be programmed with a voltage ranging between -20V to 30V.

### Example

The following example initializes the board, returns the rail connecting Channel 12 of Group A, and returns DAC B voltage:

```
SHORT nRail, nStatus;

DOUBLE dVoltage;

Gx3348Initialize (1, &nHandle, &nStatus);
Gx3348GetChannelRail(nHandle, GX3348_GROUP_A, 12, &nRail, &nStatus);

Gx3348GetDac(nHandle, GX3348_DAC_RAIL_B, &dVoltage, &nStatus);
if(nRail== GX3348_RAIL_B)
    printf("Channel 12 in Group A is connected to DAC B in Rail B");
See Gx3348SetRailSource API for a comprehensive C example.
```

### See Also

**Gx3348SetChannelRail**, **Gx3348GetDac**, **Gx3348SetDigitalOutputs**, **Gx3348SetRailSource**, **GxPdoGetErrorString**

## Gx3348SetRailSource

---

### Purpose

Sets the specified rail's source.

### Syntax

**Gx3348SetRailSource** (*nHandle*, *nRail*, *nRailSource*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>nRail</i>	SHORT	Rail number: 0. GX3348_RAIL_A: connect to rail A 1. GX3348_RAIL_B: connect to rail B 2. GX3348_RAIL_C: connect to rail C 3. GX3348_GROUND: connect to ground -1 GX3348_RAIL_NONE: no rail is connected.
<i>nRailSource</i>	SHORT	Rail source: 0. GX3348_RAIL_SOURCE_INTERNAL_DAC: Internal rail dac. All three on-board rails (A through C), have a dedicated DAC that can be set from -20V to +32V. 1. GX3348_RAIL_SOURCE_EXTERNAL rails A and C can be connected to an external source via the front J6 68 pin connector, 2. GX3348_RAIL_SOURCE_INTERNAL_5V: Rail B can be connected to an on-board 5V source. 3. GX3348_RAIL_SOURCE_EXTERNAL_AMPLIFIED: Rail A can be conneted to an external source which will be amplified by a nominal gain of 6.6 and can amplify signals from DC to > 10 KHz.
<i>PnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Rails sources are as follow:

GX3348\_RAIL\_A:

2. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail A dac can be set from -20V to +32V.
3. GX3348\_RAIL\_SOURCE\_EXTERNAL: rail A can be connected to an externla source via the front J6 68 pin connector,
4. GX3348\_RAIL\_SOURCE\_EXTERNAL\_AMPLIFIED: rail A can be conneted to an external source which will be amplified by a factor of 3.

GX3348\_RAIL\_B:

1. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail B dac can be set from -20V to +32V.
3. GX3348\_RAIL\_SOURCE\_INTERNAL\_5V: Rail B can be connected to an on-board 5V source.

**GX3348\_RAIL\_C:**

2. GX3348\_RAIL\_SOURCE\_INTERNAL\_DAC: dedicated Rail C dac can be set from -20V to +32V.
3. GX3348\_RAIL\_SOURCE\_EXTERNAL rail C can be connected to an external source via the front J6 68 pin connector

**Example**

The following example does the following:

1. Sets Rail A DAC (GX3348\_RAIL\_DAC\_A) to 5V and read back the settings.
2. Sets Rail B DAC (GX3348\_RAIL\_DAC\_B) to 10V and read back the settings.
3. Sets Rail C DAC (GX3348\_RAIL\_DAC\_A) to -5V and read back the settings.
4. Sets rail A source to its internal DAC (DAC A) and read back the settings.
5. Sets rail B source to its on-board 5V and read back the settings.
6. Sets rail C source to external and read back the settings.
7. Connect group A channel 1 to rail A and read back the settings.
8. Connect group B channel 5 to rail C and read back the settings.
9. Connect group C channel 10 to rail B and read back the settings.
10. Measure the output voltages on group A channel 1 and group C channel 10.

```
SHORT nRail, nRailsource, nStatus;
DOUBLE dVoltage, dMeasurement;
```

```
Gx3348SetDac(nHandle, GX3348_DAC_RAIL_A, 5.0, &nStatus);
```

```
Gx3348GetDac(nHandle, GX3348_DAC_RAIL_A, &dVoltage, &nStatus);
```

```
Gx3348SetDac(nHandle, GX3348_DAC_RAIL_B, 10.0, &nStatus);
```

```
Gx3348GetDac(nHandle, GX3348_DAC_RAIL_B, &dVoltage, &nStatus);
```

```
Gx3348SetDac(nHandle, GX3348_DAC_RAIL_C, -5.0, &nStatus);
```

```
Gx3348GetDac(nHandle, GX3348_DAC_RAIL_C, &dVoltage, &nStatus);
```

```
Gx3348SetRailSource(nHandle, GX3348_RAIL_A, GX3348_RAIL_SOURCE_INTERNAL_DAC, &nStatus);
```

```
Gx3348GetRailSource(nHandle, GX3348_RAIL_A, &nRailSource, &nStatus);
```

```
Gx3348SetRailSource(nHandle, GX3348_RAIL_B, GX3348_RAIL_SOURCE_INTERNAL_5V, &nStatus);
```

```
Gx3348GetRailSource(nHandle, GX3348_RAIL_B, &nRailSource, &nStatus);
```

```
Gx3348SetRailSource(nHandle, GX3348_RAIL_C, GX3348_RAIL_SOURCE_EXTERNAL, &nStatus);
```

```
Gx3348GetRailSource(nHandle, GX3348_RAIL_C, &nRailSource, &nStatus);
```

```
Gx3348SetChannelRail(nHandle, GX3348_GROUP_A, 1, GX3348_RAIL_A, &nStatus);
```

```
Gx3348GetChannelRail(nHandle, GX3348_GROUP_A, 1, &nRail, &nStatus);
```

```
Gx3348SetChannelRail(nHandle, GX3348_GROUP_B, 5, GX3348_RAIL_C, &nStatus);
```

```
Gx3348GetChannelRail(nHandle, GX3348_GROUP_B, 5, &nRail, &nStatus);
```

```
Gx3348SetChannelRail(nHandle, GX3348_GROUP_C, 10, GX3348_RAIL_B, &nStatus);
```

```
Gx3348GetChannelRail(nHandle, GX3348_GROUP_C, 10, &nRail, &nStatus);
```

```
Gx3348Measure(nHandle, GX3348_GROUP_A, 1, &dMeasurement, &nStatus);
```

```
Gx3348Measure(nHandle, GX3348_GROUP_B, 5, &dMeasurement, &nStatus);
```

```
Gx3348Measure(nHandle, GX3348_GROUP_C, 10, &dMeasurement, &nStatus);
```

**See Also**

**Gx3348SetChannelRail, Gx3348SetDac, Gx3348SetDigitalOutputs, Gx3348GetRailSource, GxPdoGetErrorString**

## Gx3348SetDigitalOutputs

---

### Purpose

Sets the eight digital output lines output enables and values.

### Syntax

**Gx3348SetDigitalOutputs** (*nHandle*, *dwOutputsEnabled*, *dwData*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nHandle</i>	SHORT	Handle to a GX3348 board.
<i>dwOutputsEnabled</i>	DWORD	Bits 0-7 represent the 8 digital outputs lines inputs. Bit zero corresponds to digital I/O line 0. Bit high will enable the specified digital output. Bit low will disable the specified digital output (default after reset).
<i>dwData</i>	DWORD	Bits 0-7 represent the 8 digital inputs. Bit zero corresponds to digital I/O line 0. Bit high will output logic high. Bit low will output a logic low (default after reset).
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

Each of the 8 Output digital lines is also connected to an input line. If the output line is not enabled, then an external digital input can be monitored using this function. Otherwise if the line output is enabled, the function returns the output line logical state.

### Example

The following example sets the lower 4 digital outputs lines to hi and the upper 4 lines to low, all odd lines (1, 3, etc) outputs are then enabled. The digital lines actual states are then read back.

```
SHORT nStatus;
DWORD dwOutputsEnabled, dwData;
DWORD dwDigitalInputs.

Gx3348SetDigitalOutputs (nHandle, 0x55, 0x0F, &nStatus);
Gx3348GetDigitalOutputs (nHandle, &dwOutputsEnabled, &dwData, &nStatus);
Gx3348GetDigitalInputs (nHandle, &dwDigitalInputs, &nStatus);
```

### See Also

**Gx3348GetDigitalOutputs**, **Gx3348GetDigitalInputs**, **GxPdoGetErrorString**



## GxPdoGetDriverSummary

---

### Purpose

Returns the driver description string and version number.

### Syntax

**GxPdoGetDriverSummary** (*pszSummary*, *nSummaryMaxLen*, *pdwVersion*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>pszSummary</i>	PSTR	Buffer to receive the summary string.
<i>nSummaryMaxLen</i>	SHORT	Buffer size passed by pszSummary.
<i>pdwVersion</i>	LPDWORD	Driver version
<i>pnStatus</i>	LPSHORT	Returned status: 0 on success, negative number on failure.

### Example

The following example returns the driver summary:

```
SHORT nHandle, nStatus;
DWORD dwVersion;
CHAR szSummary[128];
```

```
GxPdoGetDriverSummary(szSummary, 128, &dwVersion, &nStatus);
```

After the function call the parameter SzSummary will be set to:

```
"GXPDO Driver for GX3348, Version 1.00, Copyright(c) 2012 Marvin Test Solutions - MTS inc."
```

### See Also

**GxPdoGetErrorString**

## GxPdoGetErrorString

---

### Purpose

Returns the error string associated with the specified error number.

### Syntax

**GxPdoGetErrorString** (*nError*, *pszMsg*, *nErrorMaxLen*, *pnStatus*)

### Parameters

Name	Type	Comments
<i>nError</i>	SHORT	Error number.
<i>pszMsg</i>	PSTR	Buffer to the returned error string.
<i>nErrorMaxLen</i>	SHORT	The size of the error string buffer.
<i>pnStatus</i>	PSHORT	Returned status: 0 on success, negative number on failure.

### Comments

The function returns the error string associated with the *nError* as returned from other driver functions by the *pnStatus* parameter.

The following table displays the possible error values; not all errors apply to this board type:

### Warnings

- 12 Measurement Value is out of range
- 11 Unable to find suitable filter value AutoFilter mode
- 10 Measurement timeout, no signal was detected. Timeout value needs to be greater than Gate Time.

### Resource Errors

- 0 No error has occurred
- 1 Unable to open the HW driver. Check if HW is properly installed
- 2 Board does not exist in this slot/base address
- 3 Different board exist in the specified PCI slot/base address
- 4 PCI slot not configured properly. You may configure using the PciExplorer from the Windows Control Panel
- 5 Unable to register the PCI device
- 6 Unable to allocate system resource for the device
- 7 Unable to allocate memory
- 8 Unable to create panel
- 9 Unable to create Windows timer
- 10 Bad or Wrong board EEPROM
- 11 Not in calibration mode
- 12 Board is not calibrated
- 13 Function is not supported by the specified board

**General Parameter Errors**

- 20 Invalid or unknown error number
- 21 Invalid parameter
- 22 Illegal slot number
- 23 Illegal board handle
- 24 Illegal string length
- 25 Illegal operation mode

**VISA Errors**

- 30 Unable to Load VISA32/64.DLL, make sure VISA library is installed
- 31 Unable to open default VISA resource manager, make sure VISA is properly installed
- 32 Unable to open the specified VISA resource
- 33 VISA viGetAttribute error
- 34 VISA viInXX error
- 35 VISA ViMapAddress error

**Misc Errors**

- 38 Unable to lock a board resource, resource is used by another process or thread

**Parameter Errors**

- 40 Invalid Channel number (0 or 1)
- 41 Invalid Channel rail
- 42 Invalid DAC
- 43 Invalid DAC Voltage
- 44 Invalid Rail
- 45 Invalid Rail Source
- 46 Invalid Group
- 47 Invalid Gain Value
- 48 Invalid Offset Value
- 49 Invalid Calibration Data Source
- 50 EEPROM Busy Timeout

**Board/execution errors**

- 60 Unable to Set DAC

**Calibration errors**

- 80 Calibration by user, sequence error
- 81 Calibration by user, invalid calibration data
- 82 Calibration by user, invalid load value
- 83 Calibration by user, change calibration source

**Example**

The following example initializes the board. If the initialization failed, the following error string is printed:

```
CHAR    sz[256];
SHORT   nStatus, nHandle;
Gx3348Initialize (3, &Handle, &Status);
if (nStatus<0)
{   GxPdoGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz); // prints the error string returns
}
```

# Index

- .
- .NET ..... ii
- 6**
- 6U PXI.....8
- A**
- Accessories ..... 21
- ADC Calibration.....42
- Applications.....3
- Architecture ..... 1, 6
- ATEasy ..... ii, 23, 24, 25, 27, 28
- B**
- Before you Begin..... 19
- Block Diagram.....6
- Board Description..... 1, 4
- Board Handle.....30
- Board Installation ..... 19
- Borland ..... 23, 24, 28
- Borland-Delphi ..... 28
- C**
- C# ..... 24, 27
- C/C++ ..... 23, 24, 27
- C++ ..... 24, 27
- CalEasy.....ii, 39, 42
- Calibration ..... 1, 39, 44, 53, 54, 75
- Calibration Licensing ..... 39
- Calibration Procedure ..... 42
- Connector ..... 22
- Connectors ..... 21, 22
- Connectors and Accessories ..... 21
- Copyright ..... i
- Corrupt files..... 25
- D**
- DAC Voltage ..... 42
- Delphi ..... ii, 23, 27, 28
- Disclaimer..... i
- Distributing ..... 31
- Driver
  - Directory ..... 23
  - Files ..... 23
- Driver Version ..... 30
- E**
- Electric Static Discharge..... 19
- Error Handling ..... 30
- Error-Handling ..... 30
- Example ..... 24
- F**
- Features..... 3
- Final Calibration ..... 42
- Folders ..... 23
- Functions List ..... 44
- Functions Reference ..... 43
- G**
- Getting Started ..... 15
- Group A-D..... 12
- GtLinux..... 16
- Gx3348CalAdc ..... 45
- Gx3348CalDac ..... 47
- Gx3348CalSetDacVoltagePoint ..... 48
- Gx3348CalSetMode ..... 50
- Gx3348CalWriteEEPROM..... 51
- Gx3348GeChannelRail..... 55
- Gx3348GetBoardSummary ..... 52
- Gx3348GetCalibrationInfo ..... 53
- Gx3348GetDac ..... 56
- Gx3348GetDigitalInputs ..... 59
- Gx3348GetDigitalOutputs ..... 60
- Gx3348GetRailSource ..... 57
- Gx3348Initialize29, 30, 43, 44, 46, 47, 48, 49, 50, 51, 52, 61, 63, 64, 65, 66, 76
- Gx3348InitializeVisa ..... 29, 30, 44, 62

Gx3348Measure.....	63	GXPDO.PANEL.DLL.....	23
Gx3348Panel .....	64	GXPDO.PANEL.EXE.....	30
Gx3348Reset .....	30, 65	GXPDO.PANEL64.EXE.....	30
Gx3348SeChannelRail .....	67	<b>H</b>	
Gx3348SelfTest.....	66	Handle .....	19, 20, 29, 30
Gx3348SetDac.....	68	Hardware Requirements .....	39
Gx3348SetDigitalOutputs.....	72	Help .....	i
Gx3348SetRailSource .....	69	HW .....	20, 23, 27, 31
GxCntInitialize .....	10	<b>I</b>	
GxCntInitializeVisa .....	10	Initial Calibration.....	42
GxDmmInitialize .....	41	Installation and Connections.....	15
GxDmmSetCalibrationMeasurements ....	41	Installation Folders .....	23
GxDmmSetCalibrationSet .....	41	Installation: .....	19, 21
GxDmmWriteCalibrationEEPROM .	41, 42	Installing a Board.....	19
GXPDO .....	1, 16	Interface Files .....	23
Driver-Description .....	27	Introduction .....	1, 3, 39, 43
Header-file .....	27	<b>J</b>	
Help-File-Description .....	24	J3 .....	5
Panel-File-Description.....	23	J6 .....	22
GXPDO Driver .....	27	<b>L</b>	
GXPDO Driver Files Description.....	23	LabView .....	ii, 15, 16, 24, 28
GXPDO Functions.....	41	LabView/Real Time .....	28
GXPDO Software .....	15, 17	Linux .....	28
GXPDO.DLL.....	17, 23, 27, 28	<b>M</b>	
GXPDO.EXE.....	16	Measure .....	12
GXPDO.H .....	23, 27	Microsoft Visual Basic .....	24
GXPDO.LIB .....	23, 27	<b>N</b>	
GXPDO.lib .....	28	<i>nHandle</i> .....	28, 30
GXPDO.PAS .....	23, 28	<b>O</b>	
GXPDO.VB .....	23	OnError.....	28
GXPDO64.DLL.....	17, 23, 27	On-line.....	24
GXPDO64.LIB .....	23, 27	Output Channels .....	7
GXPDOBC.LIB.....	23	Overview .....	3
GxPdoGetDriverSummary .....	29, 30, 73	<b>P</b>	
GxPdoGetErrorString .....	30, 43, 74	Packing List.....	15
GxPdoInitialize.....	30	Panel .....	13, 23, 25, 29, 30, 64
GXPDO.PANEL64L.EXE.....	23	Pascal.....	23, 27, 28

PCI.....	23	Setup .....	16, 23, 25
Plug & Play.....	20	Setup Maintenance .....	25
<i>pnStatus</i> .....	30, 43	Setup-and-Installation.....	15
Program-File-Descriptions .....	23	Slot.....	10, 19, 21, 29
Programming		Specifications .....	1, 8
Borland-Delphi .....	28	Static Digital I/O.....	12
Error-Handling.....	30	Supported-Development-Tools.....	27
Panel-Program .....	30	System	
Programming the Board.....	27	Directory .....	23
Programming Using Visual Basic .....	27	System Requirements .....	15
Programming Using Visual c#.....	27	<b>T</b>	
PXI.....	3, 18, 19, 20, 21, 29	Trademarks .....	ii
PXI/PCI Explorer	10, 17, 18, 29, 30, 61, 62	Typical Output Channel Diagram.....	7
PXIeSYS.INI .....	10	<b>U</b>	
PXISYS.INI.....	10	Unpacking and Inspection .....	15
<b>R</b>		Using the GXPDO driver functions.....	29
Rail A Settings.....	11	<b>V</b>	
Rail B Settings .....	11	Virtual Panel.....	9, 10, 13, 16, 23, 29, 64
Rail C Settings .....	11	About Page .....	13
ReadMe.....	24	Initialize Dialog .....	10
README.TXT .....	23, 24	Virtual Panel Description .....	1
Readme-File .....	24	Virtual Panel Setup Page .....	11
Removing a Board.....	21	VISA. 10, 17, 18, 27, 29, 30, 44, 61, 62, 75	
Reset .....	30	Visual Basic.....	ii, 27
<b>S</b>		Visual C#.....	24
Safety and Handling .....	i	Visual C++ .....	ii, 23, 27
Sample .....	31, 32	<b>W</b>	
Sample Programs.....	31	Warranty .....	i

