The PlexDO Digital Output API
7/18/2022

Overview

PlexDO is a Plexon client API, provided in both C/C++ and Matlab versions, which allows a client program to control one or more National Instruments (NI) cards to perform basic digital output functions, such as setting individual bits high or low, outputting pulses, and generating clock signals. PlexDO can be used with or without a Plexon OmniPlex or MAP system, although when PlexDO is used to control a NIDAQ card which is also being used for MAP continuous recording, additional considerations apply.

A Win32 console mode sample program, PlexDOSampleWin32, shows how to use PlexDO for common digital output operations in a C or C++ program. You should consult the files PlexDOSampleWin32.cpp and PlexDO.h, along with this document.

Matlab sample programs, TestDOBasic.m, TestDOClock.m, TestDOStrobed1.m, TestDOStrobed2.m, and TestDOVT.m show how to use PlexDO in a Matlab program. You should consult these files, along with the .m files for each Matlab PlexDO function, e.g. PL_DOSetBit.m, along with this document. See the section "PlexDO and Matlab" for details. The Matlab code can be found in the Matlab Client SDK, available on the Plexon web site plexon.com.

Please report any problems, or direct any questions, to support@plexon.com

Hardware

PlexDO supports several NIDAQ cards and USB devices which are included with OmniPlex and MAP systems, including the following:

PCI-MIO-16E-1
PCI-MIO-16E-4
PCI-6071E
PXI-6071E
PCI-6052E
PXI-6052E
PCI-6602
PXI-6602
PCI-6503
USB-6501
PCI-6224
PXI-6224
PXI-6259
PXI-6254
PXI-6341

If you are using a NIDAQ card which is not listed above, please contact Plexon.

Each of these cards has several digital output *bits* and output *lines*. Output bits correspond to the National Instruments "DIO" outputs; typically eight bits are available, DIO0 through DIO7. Output lines correspond to the NI "GPCTR" outputs; typically two are available, GPCTR0 and GPCTR1. The 6602 cards have 32 DIO bits and eight GPCTR lines. See the NI documentation, or contact Plexon, for details on how to connect external hardware to these outputs, for example using the Plexon or NI breakout boxes or cables. The OmniPlex User Guide includes the pinouts for the breakout board used with NIDAQ cards in OmniPlex DigiAmp systems.

Note that if you are using one or more NIDAQ cards in conjunction with a MAP to perform continuous "analog channel" recording, GPCTR0 is used by the MAP for synchronization and is therefore unavailable for digital output operations using PlexDO. This restriction only applies to those NIDAQ cards being used by the MAP. The NIDAQ device numbers used by the MAP are shown in the Options dialog of the MAP Server.

IMPORTANT (for Windows XP systems only): PlexDO internally uses the NI “DAQmx” API in order to support newer devices such as the USB-6501 which are not supported by the “Traditional” NIDAQ software. However, any device which is also being used by the MAP (see above) continues to use the older method. In order for this to be transparent to applications that call PlexDO, *your NI devices must be assigned the same device numbers in both Traditional NIDAQ and DAQmx – otherwise, PlexDO will not work correctly*. See the section *PlexDO and NIDAQ Device Numbers* for information on setting device numbers. Note that the above is an issue only for Windows XP users, since National Instruments does not support Traditional NIDAQ on Windows 7; only DAQmx can be used.

Basics

PlexDO is provided as a Win32 C static link library (.lib). You need the following files to build and run an app that uses PlexDO:

```
PlexDO.h    - header file
PlexDO.lib  - library file for linker
```

You can use the sample code provided in PlexDOSampleWin32.cpp as a guide for writing your own application. Your project should include PlexDO.h and link with PlexDO.lib. Since PlexDO is statically linked into your code, no PlexDO runtime DLL is necessary.

The names of all PlexDO functions begin with `PL_DO`. All PlexDO functions return 0 to indicate success, and -1 or other nonzero error code to indicate that an error occurred. Error codes less than -10000 are NIDAQ error codes and should not be encountered in normal operation; refer to the NIDAQ documentation for their interpretation.

Although the NIDAQ hardware usually numbers channels starting at 0 (e.g. DIO0, GPCTR0), PlexDO uses 1-based numbering for output bits and output lines, for consistency with other Plexon channel numbering.

Initialization

Before performing digital output, you must determine which NI hardware devices are present and the number of output bits and lines available on each device. The `PL_DOGetDigitalOutputInfo` function must be called before any other PlexDO function and returns information on available devices:

```
unsigned int deviceNumbers[16];
unsigned int numDigitalOutputBits[16];
unsigned int numDigitalOutputLines[16];
unsigned int numCards = PL_DOGetDigitalOutputInfo(deviceNumbers,
                        numDigitalOutputBits, numDigitalOutputLines);
```

There can be up to 16 NIDAQ devices in a system, which are assigned device numbers between 1 and 16 using the NI-MAX utility (see the NI documentation for details on using NI-MAX). Note that these device numbers are not necessarily contiguous, but if you are using Windows XP, *they must be identical in both Traditional NIDAQ and DAQmx* (see the section on PlexDO and NIDAQ Device Numbers).

Upon return from the `DOGetDigitalOutputInfo` function, the first `numCards` entries in each of the above arrays is filled in with information on the NI devices which were found. Devices which have no digital output capability, or which are not supported by PlexDO, are not included. For example, on a system containing two NI cards with digital outputs, a PCI-6071E at NIDAQ device number 2 (Dev2) and a PCI-MIO-16E-1 at NIDAQ device number 5 (Dev5), you would have:

```
numCards = 2
deviceNumbers[0] = 2
deviceNumbers[1] = 5
numDigitalOutputBits[0] = 8
numDigitalOutputBits[1] = 8
numDigitalOutputLines[0] = 2
numDigitalOutputLines[1] = 2
```

Most PlexDO functions take the NI device number of the device to be controlled as their first parameter. In the above example, you could control devices 2 and / or 5. Operations on each device are completely independent of operations on other devices; in particular, there is no synchronization of timing between multiple cards, e.g. you cannot generate a pulse at exactly the same time on two different digital output cards. If you need to output digital words with more than eight bits, consider using the NI 6602 cards, which have 32 DIO bits.

You can call the `PL_DOGetDeviceString` function to obtain a text string that describes the device with a given NI device number, e.g. if you wish to present a list of available devices to a user.

```
for (int board = 0; board < 16; board++)
  if (deviceNumbers[board] != 0)
  {
    char deviceString[64];
    PL_DOGetDeviceString(deviceNumbers[board], deviceString);
    printf("NI devicenumber=%d, model=%s, %d DO bits, %d DO lines\n",
          deviceNumbers[board], deviceString,
          numDigitalOutputBits[board], numDigitalOutputLines[board]);
  }
```

Once you have chosen the device(s) to be used, the NI device number is used to refer to it in subsequent PlexDO calls.

You must use the PL_DOInitDevice function to initialize each device before using it for digital output.

```
PL_DOInitDevice(deviceNum, false);
```

The second argument should be `false`, unless the device is also being used by the Plexon MAP.

Output Bits

Once the device has been initialized, you can use PlexDO to control the output *bits* and *lines* on the device. Each output bit can be independently set to low (logic "0," or 0V) or high (logic "1," or +5V). For example, you might want to set bit 0 (DIO0) to 1 to indicate the start of a trial in your experiment, then set it to 0 at the end of the trial. Any external hardware connected to DIO0 will then see a "1" (+5V) while the trial is in progress. Bits remain in the state you set, low or high, until you change them, i.e. they are latched.

Before manipulating the output bits on a device, you should use the `PL_DOClearAllBits` function to set all the bits to 0.

```
PL_DOClearAllBits(device);
```

Now you can use the `PL_DOSetBit` and `PL_DOClearBit` functions to set any output bit to 1 or 0 respectively. Note that bits are numbered starting at 1, although NIDAQ hardware numbering starts at 0, i.e. PlexDO bit 1 corresponds to DIO0 on most NIDAQ cards.

```
PL_DOSetBit(device, bit);
PL_DOClearBit(device, bit);
```

You can also use the `PL_DOSetWord` function to set multiple bits in one call. Specify the range of bit numbers to be set and the value to be assigned.

```
PL_DOSetWord(device, lowBit, highBit, value);
```

For example, if all output bits are zero, then the call

```
PL_DOSetWord(device, 3, 6, 0x28);
```

will result in the following:

```
0x28 = 0010 1000

8 7 6 5 4 3 2 1  (bit number)
0 0 1 0 1 0 0 0  (bit values)
```

Bits 3, 4, 5, and 6 are set to 0, 1, 0, and 1 respectively. Bits 1, 2, 7, and 8 are unaffected. Note that the value parameter is a 32 bit unsigned integer, in order to support devices with up to 32 output bits, so that the `0x28` above is actually expanded to `0x00000028`, i.e. the rightmost eight bits of the 32 bit parameter are used.

Note that a call to `PL_DOSetWord` is exactly equivalent to a series of calls to `PL_DOOSetBit` and `PL_DOClearBit`, i.e. the bit values are assigned sequentially, not all at the exact same time, although a call to `PL_DOSetWord` is faster than a series of calls to `PL_DOOSetBit` and `PL_DOClearBit`.

Besides setting and clearing bits and writing word values, you can also *pulse* an output bit. Although an output bit should not be used when you need a very short pulse (less than a millisecond) or very precisely timed pulses (e.g. 7.25 milliseconds), it can be used in many non-critical cases (for high-precision pulse generation, see the section on Output Lines below). Pulsing a bit is done with the `PL_DOPulseBit` function:

```
PL_DOPulseBit(device, bit, 1); // 1 millsecond long pulse
```

This sets the specified bit high for at least the specified duration in milliseconds, which must be an integer value. Note that this is not a hardware-timed pulse, so depending on Windows system

activity, the actual pulse length may be longer than what you requested, but it will never be shorter. Again, if you need exact pulse widths, use an output line rather than an output bit.

Using `PL_DOPulseBit`, we now have the tools to output a strobed word. A strobed word is one in which several adjacent output bits represent the actual value or code to be sent to some external hardware device, and an additional bit is the strobe bit, which is set to 1 to signal when the strobed word value is ready to be read by the external hardware. Assuming that the width of the strobe bit is not critical (this will depend upon the characteristics of the external hardware), we can use `PL_DOPulseBit` to control the strobe bit as follows:

```
PL_DOClearAllBits(device);

// write a seven-bit value to bits 1 through 7
PL_DOSetWord(device, 1, 7, value);

// pulse the strobe bit (bit 8) for at least 1 msec
PL_DOPulseBit(device, 8, 1);
```

The value of bits 1 through 7 is only read by the external hardware while bit 8, the strobe, is high. Of course, we could use any bits for the strobed word, or even define multiple independent strobed words, each with their own strobe bit. For example, If we denote two strobed words as w and x, each three bits wide:

```
ws w3 w2 w1 xs x3 x2 x1
 8  7  6  5  4  3  2  1  (bit/DIO number)
```

i.e., ws is the strobe bit for the three-bit strobed word w3, w2, w1, and xs is the strobe bit for the three-bit strobed word x3, x2, x1. If we wanted to write the three-bit value 110 for w:

```
ws w3 w2 w1 xs x3 x2 x1
 8  7  6  5  4  3  2  1  (bit/DIO number)
 0  1  1  0  0  0  0  0

 0110 0000 = 0x60

// write the three-bit value 110 to bits 5 through 7
PL_DOSetWord(device, 5, 7, 0x60);

// pulse the strobe bit (bit 8) for at least 1 msec
PL_DOPulseBit(device, 8, 1);
```

Likewise, to write a three-bit strobed value 101 for x:

```
ws w3 w2 w1 xs x3 x2 x1
 8  7  6  5  4  3  2  1  (bit number)
 0  0  0  0  0  1  0  1

 0000 0101 = 0x05

// write a three-bit value 101 to bits 1 through 3
PL_DOSetWord(device, 1, 3, 0x05);
```

```
// pulse bit 4 for at least 1 msec
PL_DOPulseBit(device, 4, 1);
```

Of course, many other combinations and encodings are possible, limited only by the number of output bits available and your external hardware and cabling. You could use the lowest five bits for a four-bit strobed word plus its strobe bit, and the upper three bits as three individual unstrobed bits, etc. PlexDO gives you the API to manipulate the bits; the semantics of how the bits are organized and used is up to you and the external hardware that reads the bits.

Output Lines

As opposed to output bits, output *lines* cannot be latched to a logic 0 or 1 value; rather, each output line is used to either output individual pulses of precise durations, or clock signals of precise frequencies. PlexDO line numbers start at 1, line 1 corresponding to the NI designation GPCTR0 on typical NIDAQ cards.

Call the `PL_DOSetLineMode` function to set the mode for each line to be used to either `PULSE_GEN` or `CLOCK_GEN` as desired, e.g.

```
// set line 2 (GPCTR1) to pulse generation mode
PL_DOSetLineMode(deviceNum, 2, PULSE_GEN);
```

Important Note: If you are using the device at the same time that a Plexon MAP system is using the device for continuous recording, the first output line (GPCTR0) is reserved by the MAP for synchronization and cannot be used by PlexDO.

Output Line Pulse Generation

If an output line is set to pulse generation mode, you can use the `PL_DOSetPulseDuration` function to set the pulse width with a resolution of of one microsecond:

```
// set a pulse width of 125 usec on line 2
PL_DOSetPulseDuration(deviceNum, 2, 125);
```

This call does not actually output any pulses, it only defines the pulse width to be used. To actually output a pulse, use the `PL_DOOutputPulse` function:

```
// outputs a single pulse on line 2
PL_DOOutputPulse(deviceNum, 2);
```

The effect is like that of `PL_DOPulseBit`, except that an output line is used instead of an output bit, and the width of the pulse is guaranteed to be as specified, with a hardware-timed accuracy of better than 50 nanoseconds, independent of Windows system activity. There is currently no support for generating a specific number of pulses, although this may be added in future versions of PlexDO.

The previous examples of strobed word output apply; simply use `PL_DOOutputPulse` instead of `PL_DOPulseBit` to pulse the strobe bit. Besides the guaranteed pulse width, this has the advantage that using an output line for the strobe bit frees up an output bit for other uses. However, keep in mind that many NIDAQ cards have only two output lines, and if a Plexon MAP is using the card, only the second line is available.

Output Line Clock Generation

If an output line is set to clock generation mode, you can set the high and low times of the clock signal with a resolution of one microsecond:

```
// set up clock on line 2 (GPCTR1): 10 usec high, 30 usec low
PL_DOSetClockParams(deviceNum, 2, 10, 30))
```

This clock signal will have a frequency of

```
1000000 / (10 + 30) = 25000 Hz = 25 kHz
```

and a duty cycle of

```
10 / (10 + 30) = 25%
```

If you need a simple 50% duty cycle clock signal of a given frequency, both the high and low times are given by:

```
tHigh = tLow = 500000 / FrequencyInHz
```

`PL_DOSetClockParams` does not actually output a clock signal but only defines its characteristics. To start clock output on a given line, use the `PL_DOStartClock` function:

```
PL_DOStartClock(deviceNum, 2); // output a clock signal on line 2
```

The clock signal will be output continuously until you stop it using `PL_DOStopClock`:

```
PL_DOStartClock(deviceNum, 2); // stop clock signal on line 2
```

To change the frequency or duty cycle of a running clock signal you must stop the clock, call `PL_DOSetClockParams` with the new settings, then start the clock again.

Remember that if the device is being used by a Plexon MAP, output line 1 (GPCTR0) is reserved and cannot be used by PlexDO.

PlexDO and Matlab

The PlexDO functions can also be called from a Matlab program. The NI hardware that is supported and the functionality are identical to the C/C++ version, only the syntax is different.

You need the following files to run a Matlab app that uses PlexDO:

```
mexPlexDO.mexw64   - PlexDO library
PL_DO*.m           - used by Matlab to interface to mexPlexDO.mexw32
```

Each .m file contains a description of that function and its parameters, similar to that provided in the PlexDO.h file for C/C++. These files are located in the Matlab Client SDK, available from the Plexon web site plexon.com. Copy the dll and the .m files to a directory that is in your Matlab path.

Three sample programs are provided to demonstrate use of PlexDO in a Matlab program:

```
TestDOBasic.m      - setting and clearing output bits
TestDOStrobed1.m   - strobed word output using a GPCTR as the strobe
TestDOStrobed2.m   - strobed word output using a DIO as the strobe
TestDOClock.m      - clock generation using a GPCTR
TestDOVT.m         - setting bits based on video tracking coordinates
```

Although the sample code and the comments in the .m files are sufficient to learn how to use PlexDO, you may also wish to read the above documentation for the C/C++ version for additional description of the functions and their use. Again, only the function syntax is different, the semantics are the same.

Note that an especially compute-intensive Matlab program can increase the latency and variability of PlexDO commands. However, GPCTR clock frequencies and pulse widths are controlled by the NI hardware and so are not affected by Windows CPU usage.

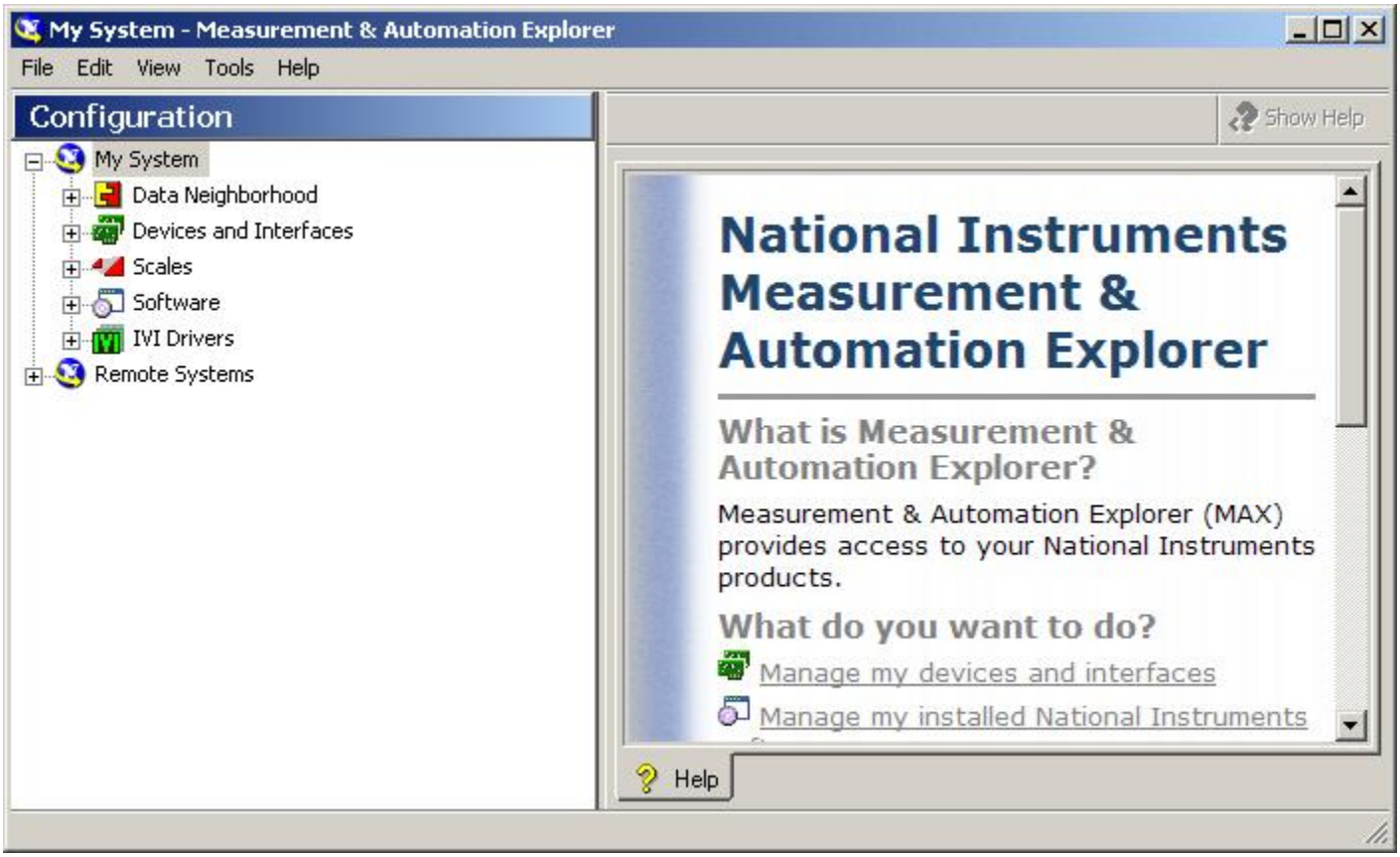
PlexDO no longer supports 32 bit versions of Matlab.

PlexDO and NIDAQ Device Numbers

The following section applies to Windows XP systems only and can be skipped by users of later versions of Windows.

In order to support newer National Instruments devices while maintaining backwards compatibility with older hardware and software, PlexDO uses both “flavors” of NI drivers, referred to as “Traditional NIDAQ” and “DAQmx.” Since Rasputin (the MAP system software) uses Traditional NIDAQ, and the same device cannot be accessed by both Traditional and DAQmx at the same time, any device being used by the MAP (as indicated by the second parameter to `PL_DOInitDevice`) is accessed using Traditional NIDAQ; otherwise, DAQmx is used. This is purely internal to PlexDO and is irrelevant to users of PlexDO, except for one important detail: the NIDAQ device numbers for a device must be the same for both Traditional

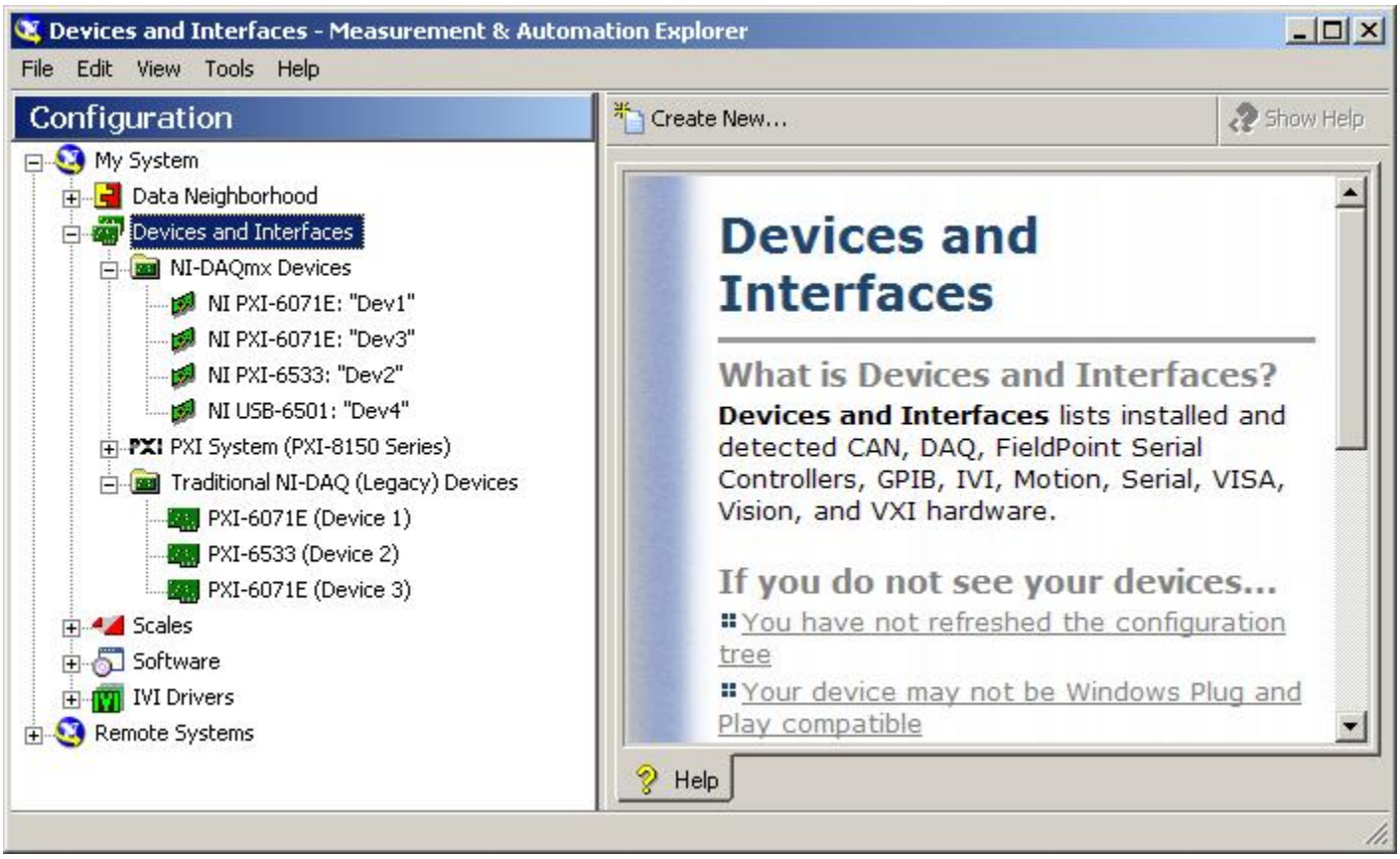
NIDAQ and DAQmx. You can use the Measurement & Automation Explorer utility (a.k.a. NI-MAX), found in the National Instruments program group, to inspect and set NIDAQ device numbers.



Expand the “Devices and Interfaces” section:

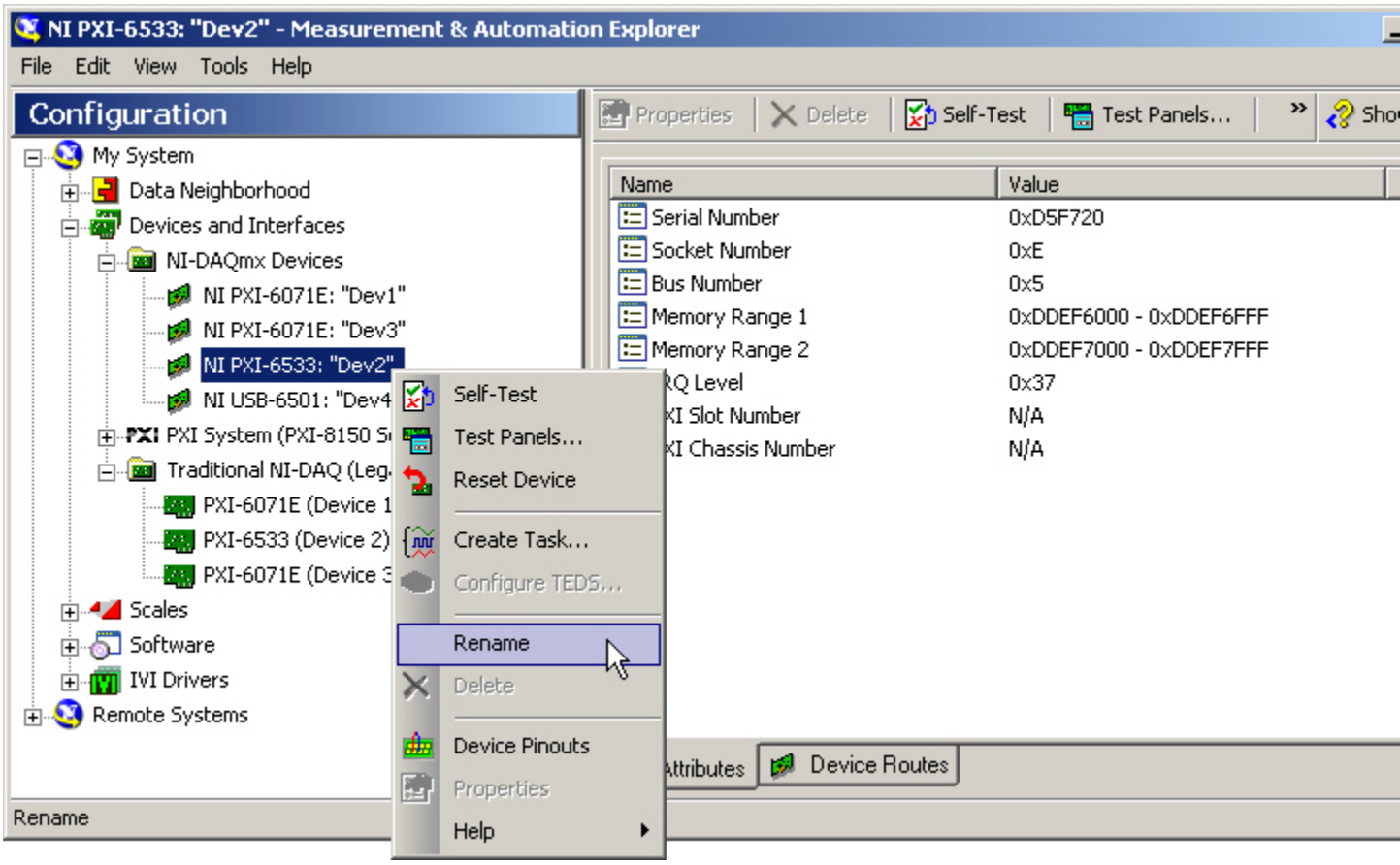
You should see both “NI-DAQmx Devices” and “Traditional NI-DAQ (Legacy) Devices” – if only one of the two appears, you must close NI-MAX and install the other subsystem from the National Instruments installation CDs before proceeding.

Next, expand both device sections:



Note that in this example, three of the four installed devices appear in both Traditional and DAQmx, but the USB-6501 only appears in DAQmx, because it is a newer device which is not supported in Traditional NIDAQ. Devices in an external rackmount chassis will begin with "PXI," devices mounted directly in a PC will begin with "PCI," and devices connected via USB will begin with "USB."

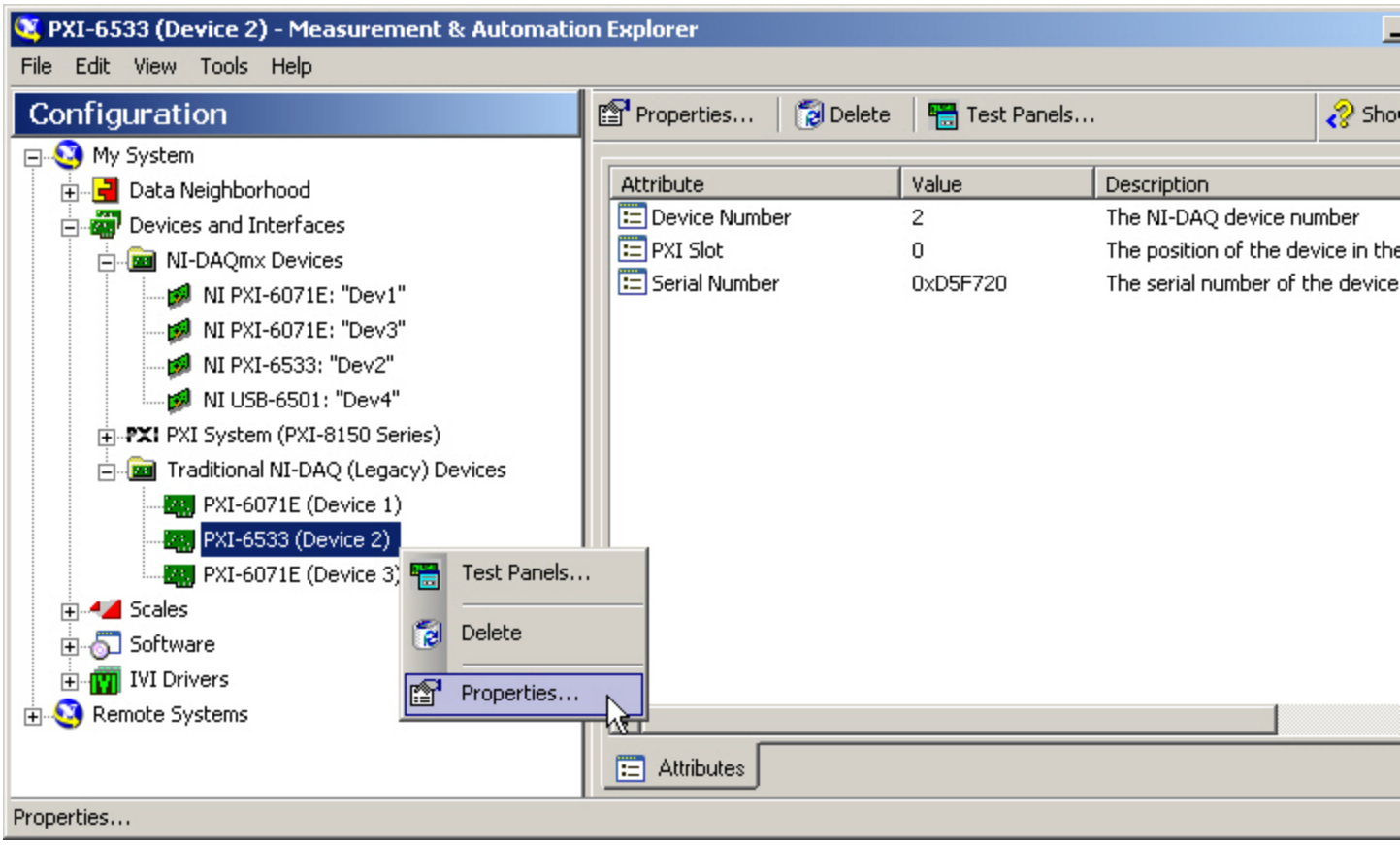
For each device which appears in both lists, make sure that the device numbers are the same for both Traditional and DAQmx. For example, for the PXI-6533, you can change its DAQmx device number by right-clicking on it and using the Rename command:



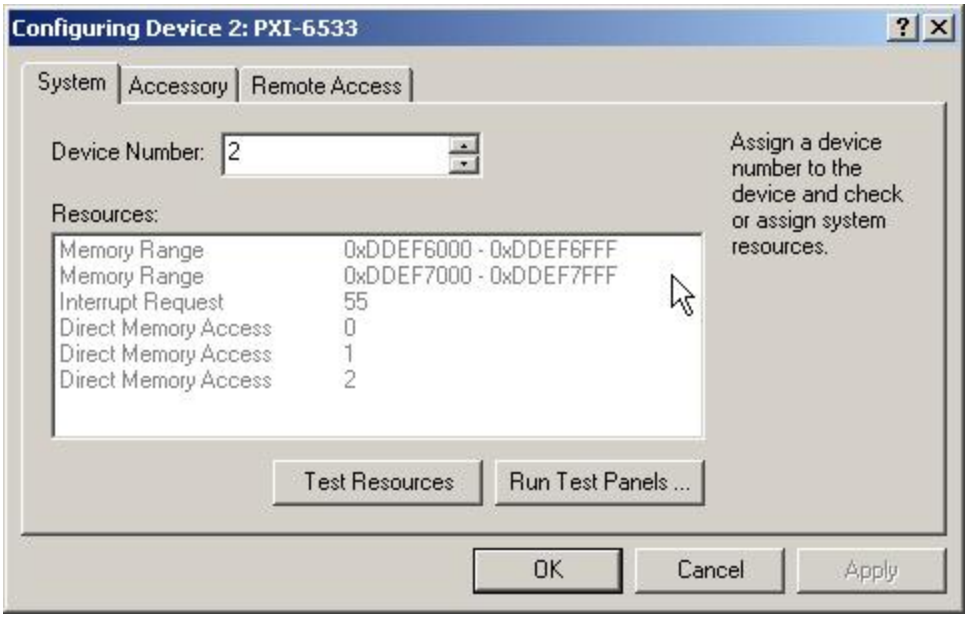
Make sure when renaming DAQmx devices to use the “Dev” prefix, e.g. “Dev2.”

IMPORTANT: For reasons described below, it is preferred to leave the Traditional NIDAQ device numbering unchanged, and change only the DAQmx device numbering so as to make the two sets of device numbers correspond. If for some reason you find it necessary to change the Traditional device numbering, you may have to update some settings in the MAP Server options as described later.

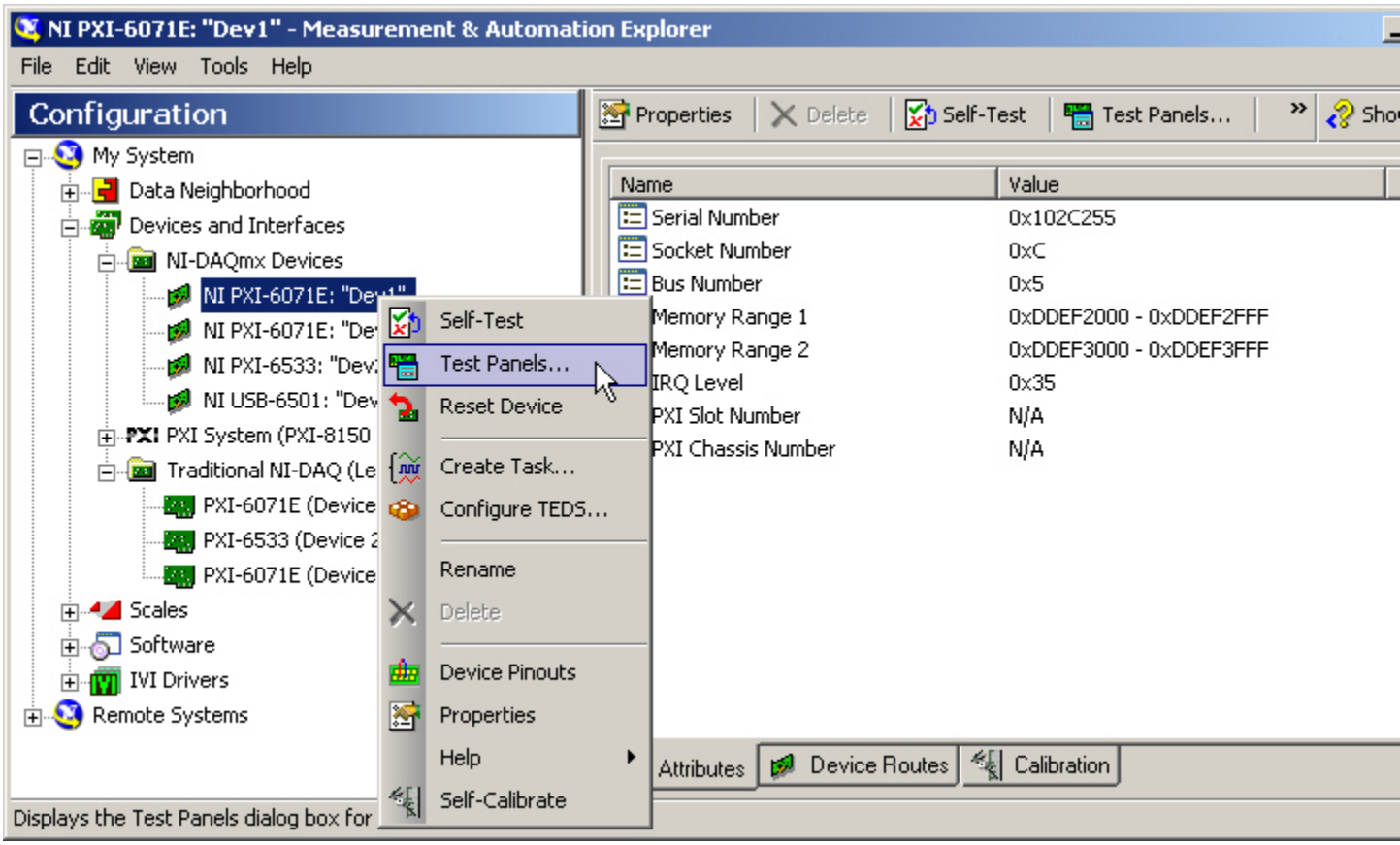
If it is necessary to rename the corresponding Traditional device, right click on its entry and select Properties:



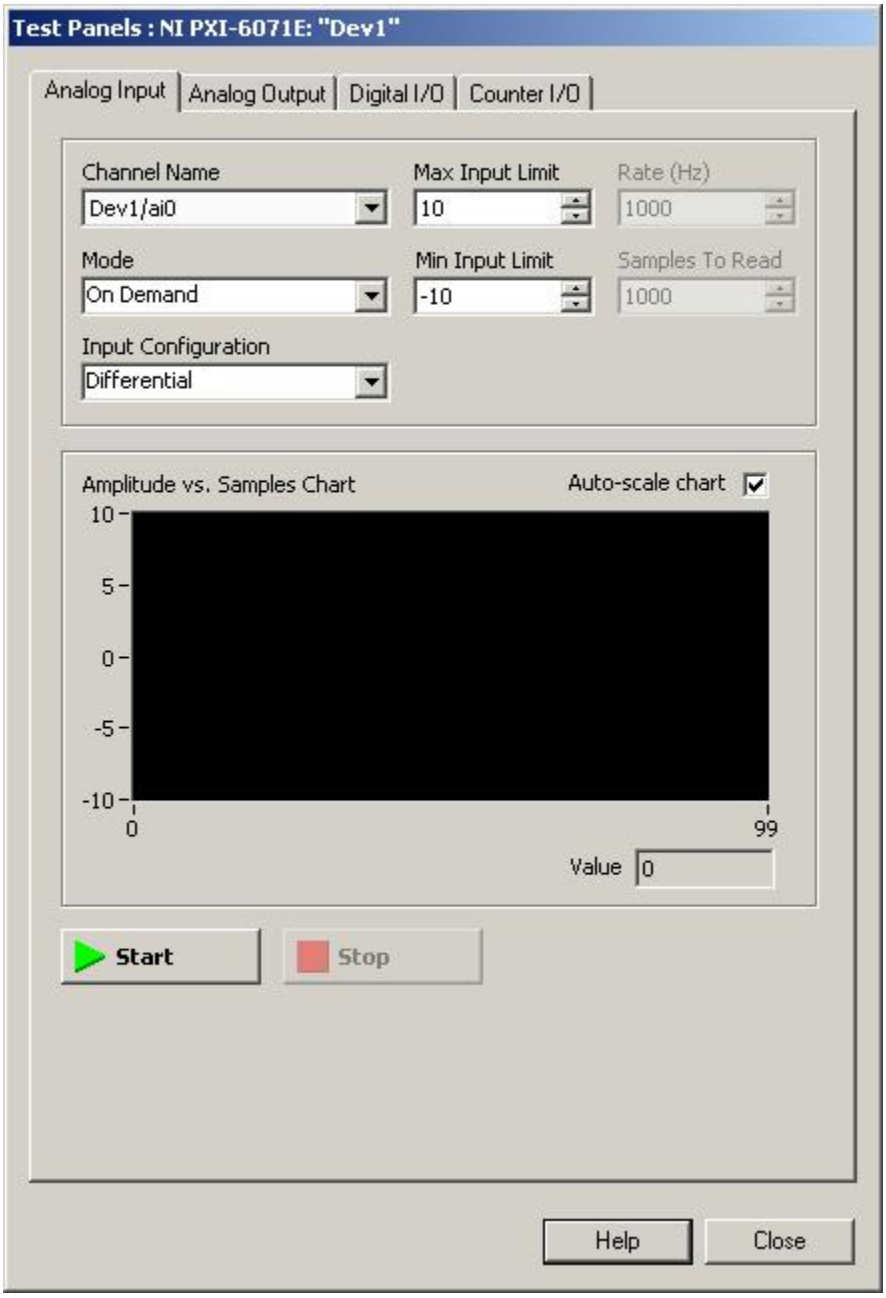
This will display the Properties dialog, which allows you to specify the device number. Here, you only enter the device number, with no prefix:



In many cases, this is all you need to do to set the device numbers in DAQmx or in Traditional. However, if you have more than one card of the same type, e.g. the two 6071E cards in the above example, you need a way of determining which physical card corresponds to a card in each list; i.e. you cannot assume that "Dev1" and "Device 1" refer to the same physical card. One way of doing this is to inject a known signal into one of the cards, and then use the Test Panel to view the signal on each card. To display the Test Panel, right click on the desired card and select Test Panels:

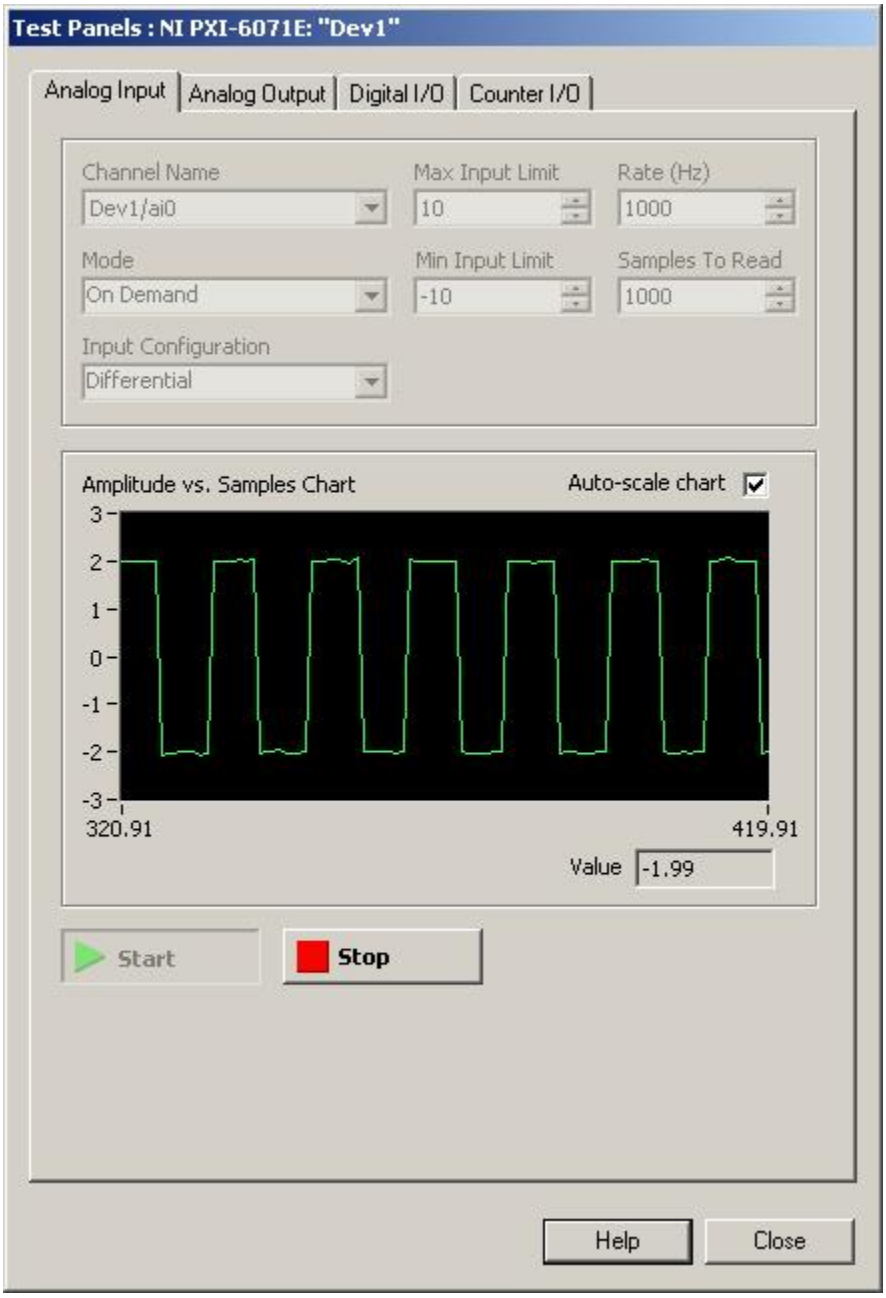


The Test Panel for the selected device is displayed:



Connect a known signal, such as the output of a function generator, to the first channel on one of the 6071E cards (or whatever card you have more than one of). The details of this will depend on your cabling, breakout boxes, etc.

Click the Start button to show the signal in NI-MAX. In this example, we used a 1 Hz, 4V p-p square wave from a function generator as the test signal:



If you do not see the signal in the Test Panel, the card whose input it is displaying is not the one you are applying the test signal to. You can either close the Test Panel and open a Test Panel for one of the *other* 6071E cards, or you can try inputting the test signal to a different card, while leaving the Test Panel open so that you can see when you have "hit" the right card. Once you have done this, you can close the Test Panel and repeat the procedure for the same card but for the other "flavor," i.e. Traditional or DAQmx. Once you have established which Traditional and DAQmx devices correspond to the same physical device, you can change their device numbers as necessary so that both Traditional and DAQmx use the same device number to refer to the same device.

The above procedure only covers cards which have A/D channels in addition to digital output channels. If you have multiple cards which perform digital I/O only, e.g. more than one PCI-6602 or USB-6501, you will need to perform similar tests, but using the Digital I/O tab in Test Panel, one of the card's digital outputs, and a scope or an LED with current-limiting resistor to monitor the digital output signal.

Note that you must insure corresponding Traditional and DAQmx device numbers for *all* NIDAQ cards in your system, not just the ones that you will use with PlexDO.

IMPORTANT: *If you change the Traditional device numbers using NI-MAX, it is possible that you will need to change the device numbers in the MAP Server options to correspond to the new numbering. In particular, the HLK2 board in the MAP communicates with a NIDAQ card, either a PCI-DIO-32HS or a PCI/PXI-6533 – if this is not set to the same Traditional device number in both NI-MAX and in the MAP Server options, the Rasputin software will be unable to "talk" to the MAP hardware. To avoid having to change the Server settings, leave the Traditional device numbers unchanged, and change only the DAQmx device numbers,*

Likewise, if the A/D cards (e.g. 6071E) are not set to the same Traditional device numbers in NI-MAX and in the Server options, you will not be able to acquire and record some or all NIDAQ continuous channels. Again, changing only the DAQmx device numbers will avoid the need to change the device numbers in Server.