Using the Direct I/O COM API of DriverLINX for the KPCI-PIO24 or KPCI-PIO96 with LabWindows/CVI

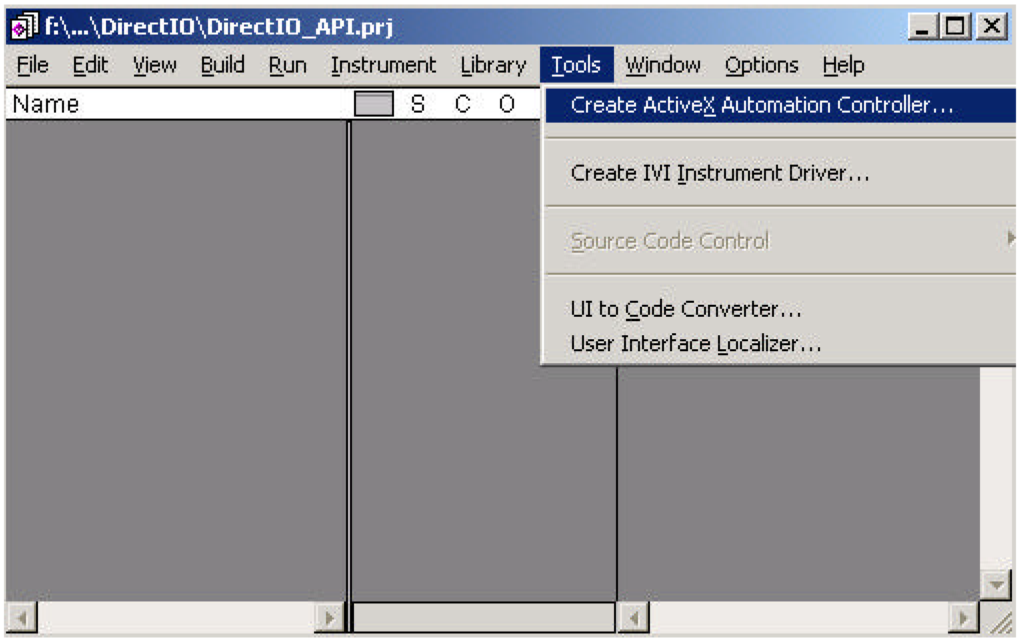
DriverLINX drivers for digital I/O boards provides two programming interfaces:

1. Service Request API for hardware independence with other KPCI or PIO boards
2. Direct I/O COM API for simple, fast interaction with the registers of the digital I/O hardware

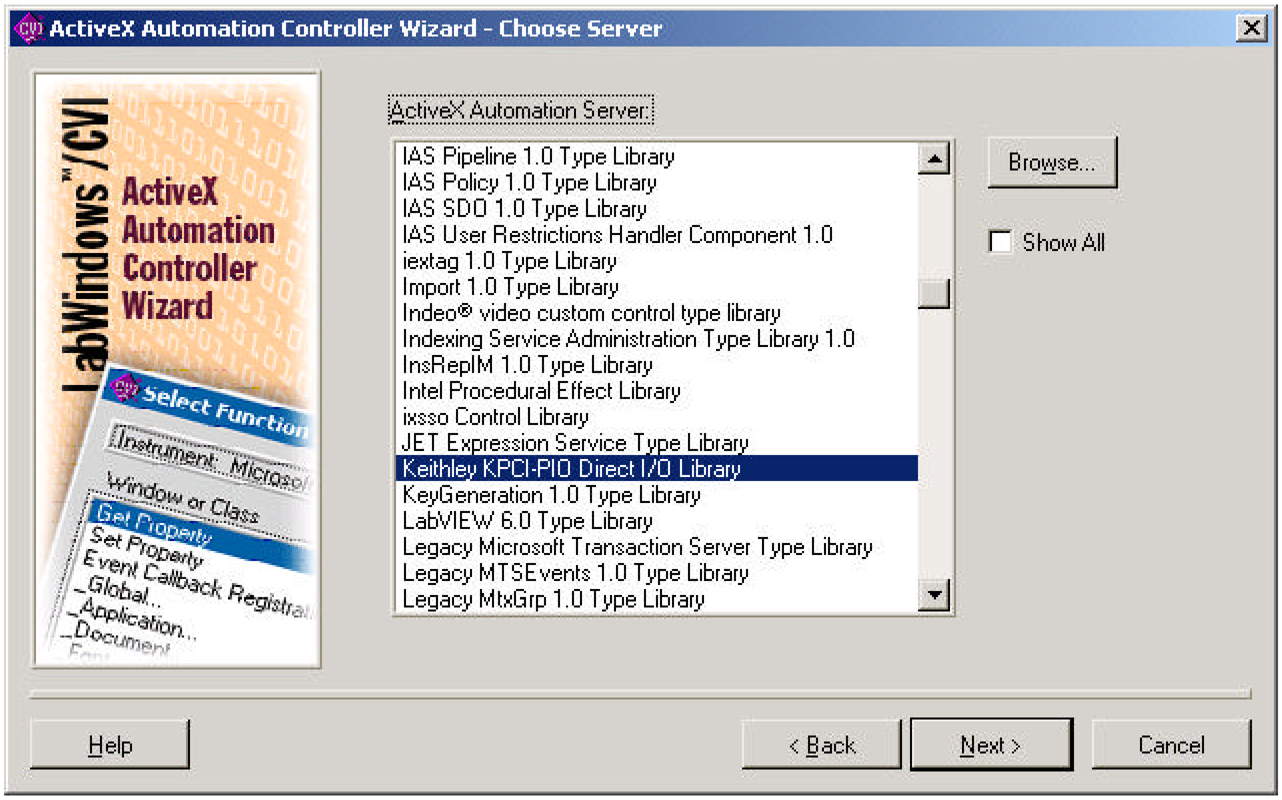
COM objects are easily used from Visual Basic or from C++. Use of a COM object from a purely C language such as CVI requires a bit more work to build a proper interface layer into the object. Luckily, LabWindows/CVI provides an automated tool to greatly simplify creation of the interface layer. The resulting function panel file, *.fp, can then be used in conventional ways by CVI to make use of the function calls of the exposed object.

Below is a step-by-step guide. The completed project is also available as an example program in the download center.

Step 1: In the Tools menu, select 'Create ActiveX Automation Controller....'

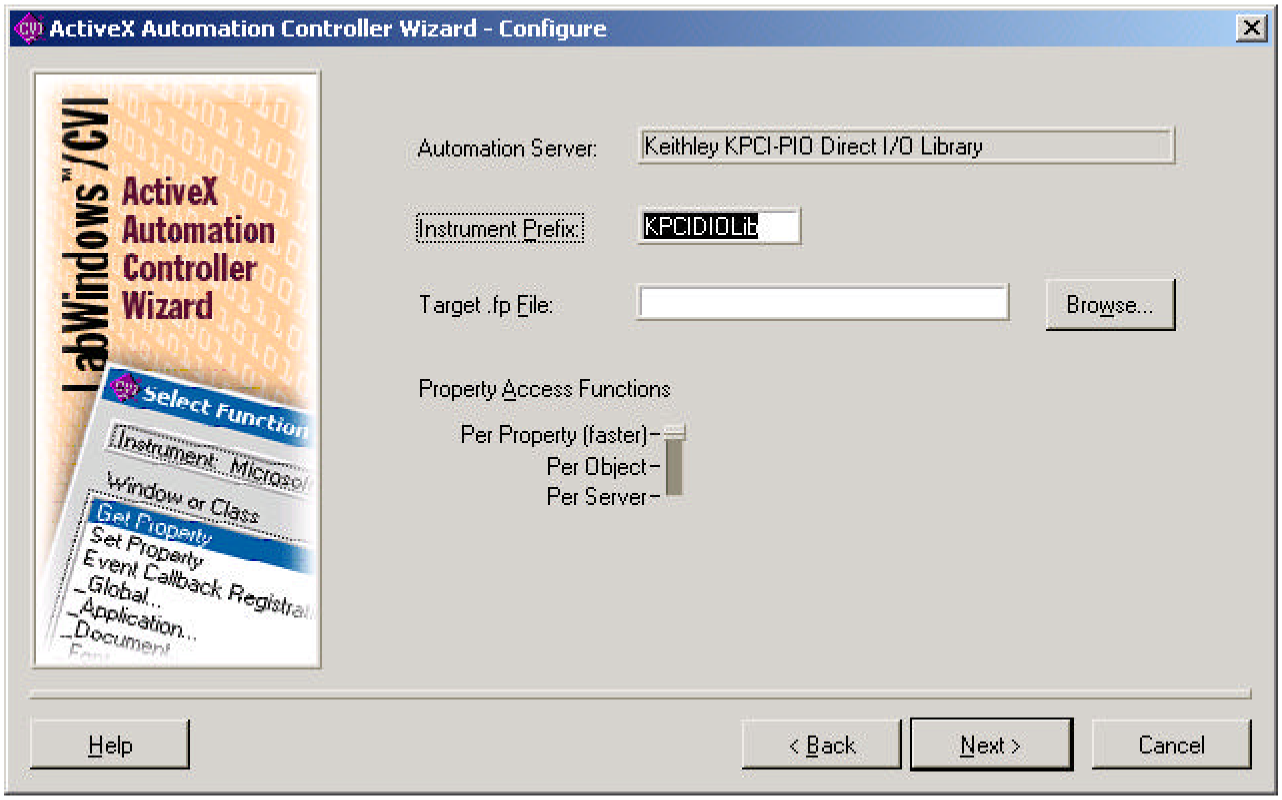

f:\...\DirectIO\DirectIO_API.prj
File
Edit
View
Build
Run
Instrument
Library
Tools
Window
Options
Help
Name
S
C
O
Create ActiveX Automation Controller...
Create IVI Instrument Driver...
Source Code Control
UI to Code Converter...
User Interface Localizer...

Step 2: Scroll down the list until you see the Keithley Digital I/O Library. Select it and then click Next.



IAS Pipeline 1.0 Type Library
IAS Policy 1.0 Type Library
IAS SDO 1.0 Type Library
IAS User Restrictions Handler Component 1.0
iextag 1.0 Type Library
Import 1.0 Type Library
Indeo® video custom control type library
Indexing Service Administration Type Library 1.0
InsReplM 1.0 Type Library
Intel Procedural Effect Library
ixsso Control Library
JET Expression Service Type Library
Keithley KPCI-PIO Direct I/O Library
KeyGeneration 1.0 Type Library
LabVIEW 6.0 Type Library
Legacy Microsoft Transaction Server Type Library
Legacy MTSEvents 1.0 Type Library
Legacy MtxGrp 1.0 Type Library

Step 3: Fill in a name for the target .fp file. Any name is fine; I used KPCIDIO (be sure to include the path to the location). Then click the Next button.



ActiveX Automation Controller Wizard - Configure

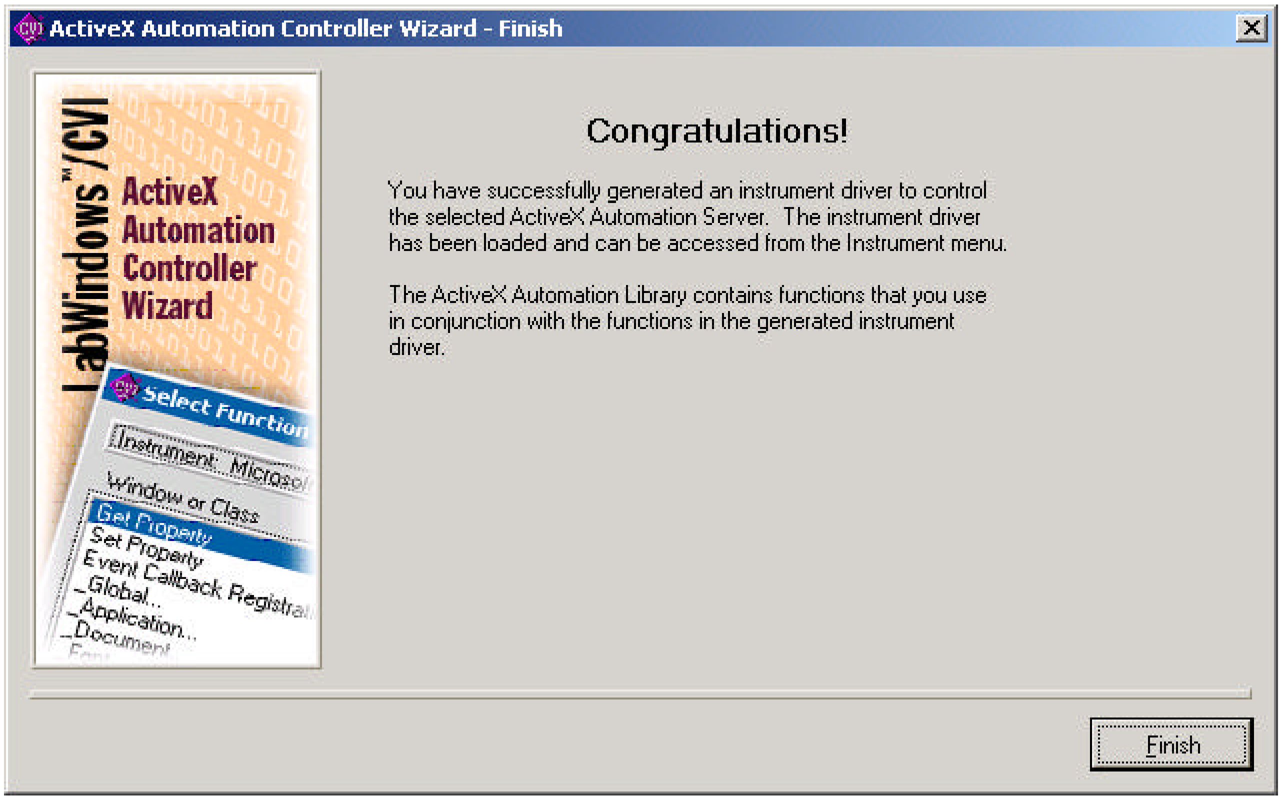
Automation Server: Keithley KPCI-PIO Direct I/O Library

Instrument Prefix: KPCIDIOLib

Target .fp File:

Property Access Functions: Per Property (faster) / Per Object / Per Server

Step 4: the object will be analyzed by CVI and several output files will be built and placed into the directory location you specified. If any other dialogs present, accept the default values and click Next. When complete, the screen below will appear.

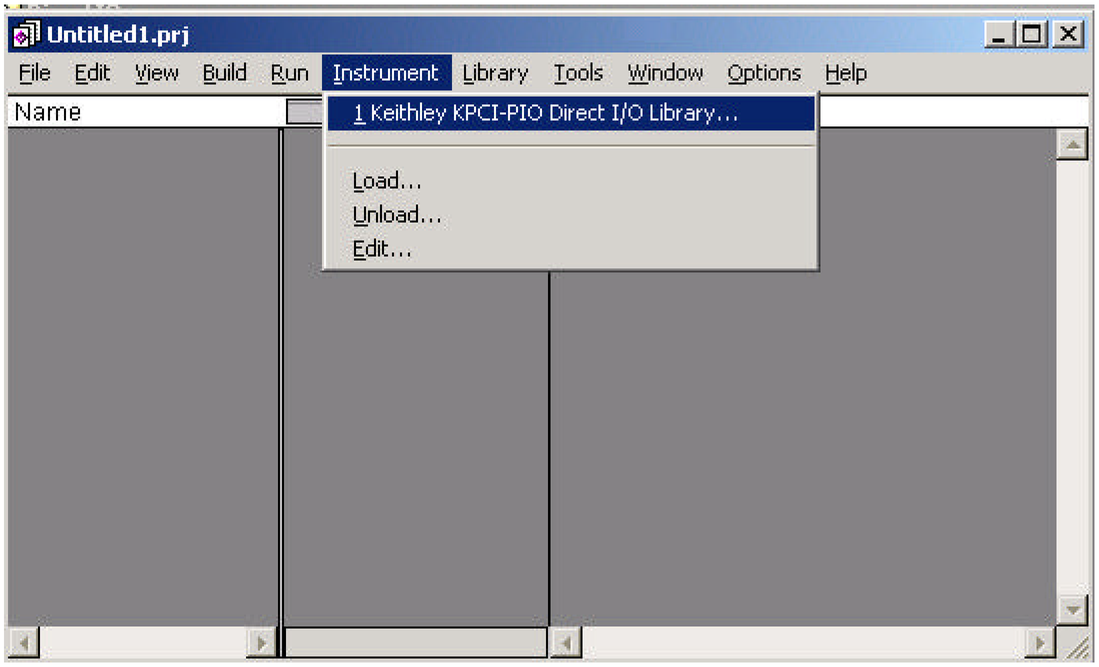


Congratulations!

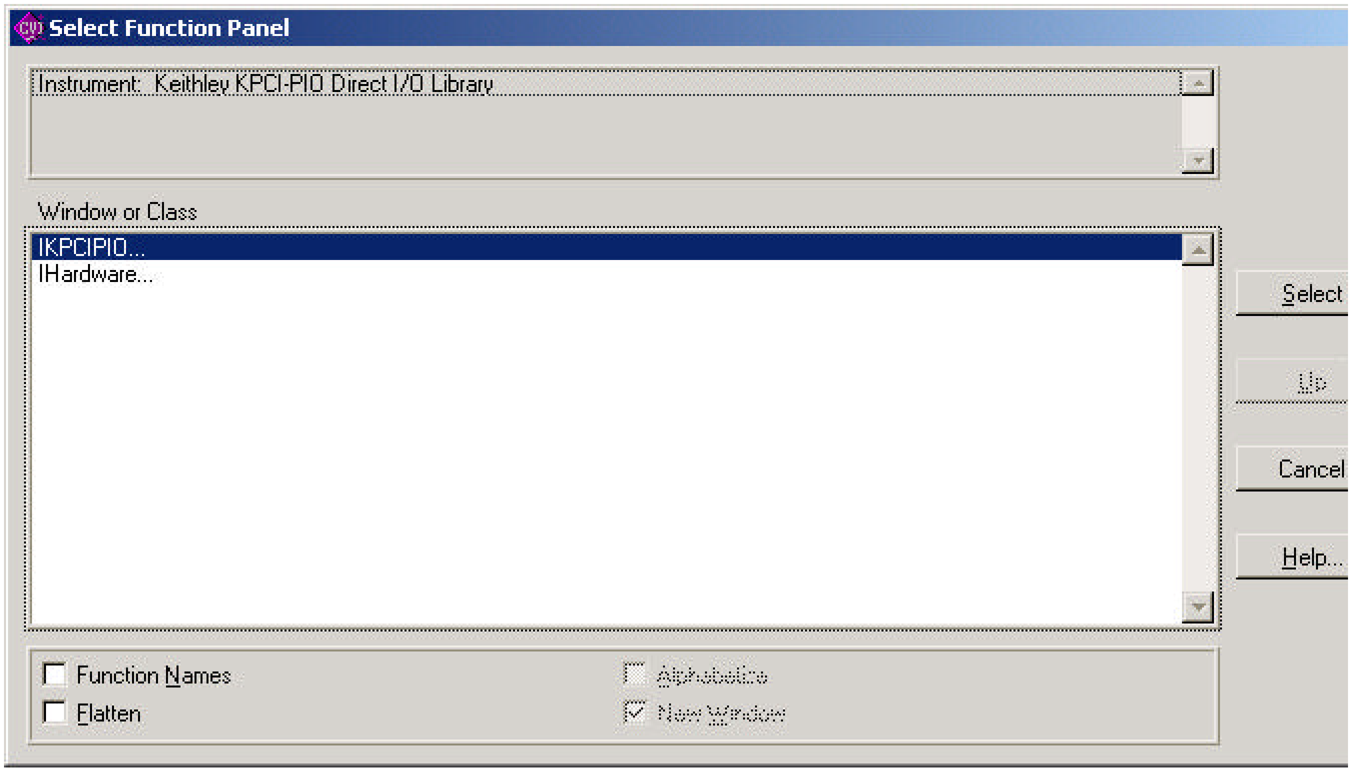
You have successfully generated an instrument driver to control the selected ActiveX Automation Server. The instrument driver has been loaded and can be accessed from the Instrument menu.

The ActiveX Automation Library contains functions that you use in conjunction with the functions in the generated instrument driver.

Step 5: Now to use the object. From this same session of CVI, the function panel will already be loaded in the Instrument Driver menu drop down. If starting a new session of CVI, first load the fp file.



Step 6: Notice the object has two classes: IKPCIPIO and IHardware. IKPCIPIO provides an 8255 interface: a control register and one register for each of three 8-bit digital ports. The IHardware class provides 32-bit register access to the KPCI-PIOxx card. The balance of this application note will refer to the IKPCIPIO class only. Highlight that class as shown below and then click the Select button.

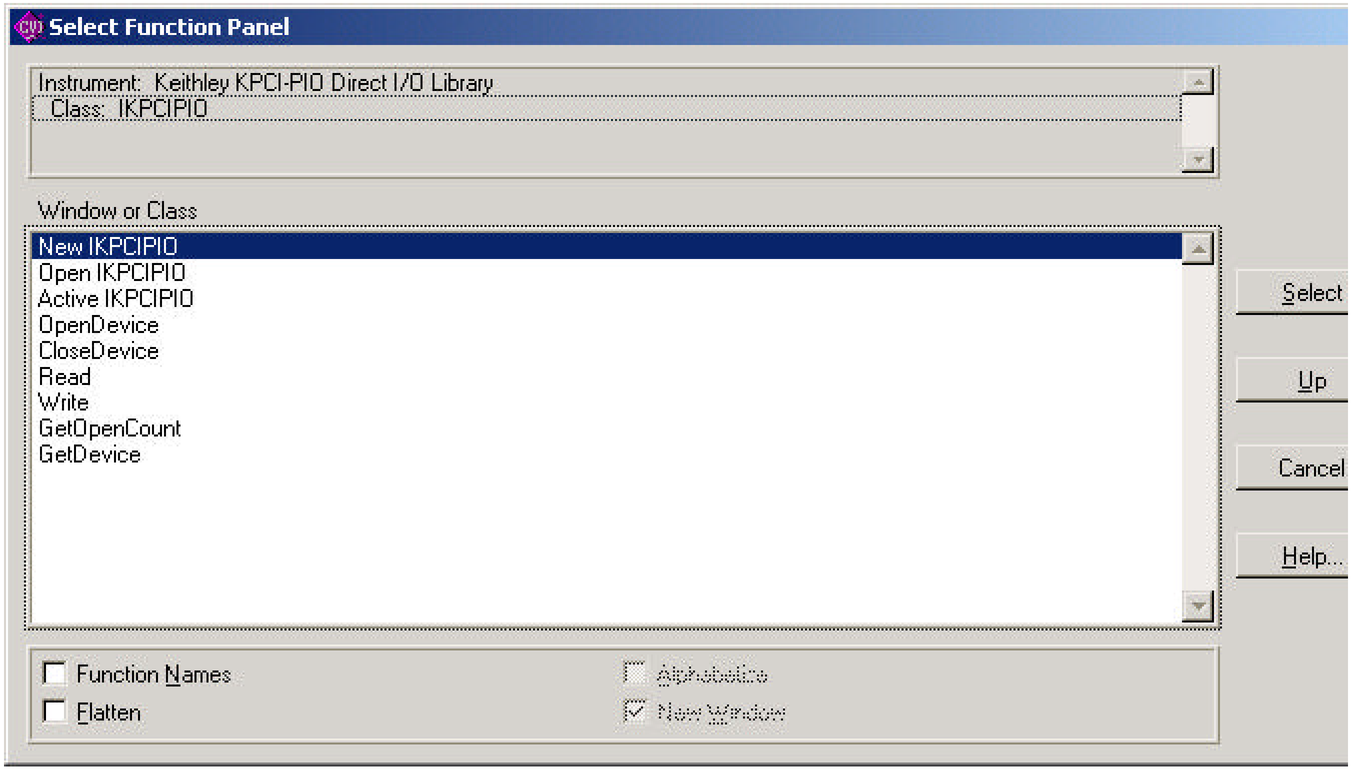

Select Function Panel

Instrument: Keithley KPCI-PIO Direct I/O Library

Window or Class

IKPCIPIO...
IHardware...

Once the IKPCIPIO class has been selected, the member functions are visible. The important ones are New, OpenDevice, CloseDevice, Read and Write. This was just a preview of where we will be going, so hit the Cancel button for now.



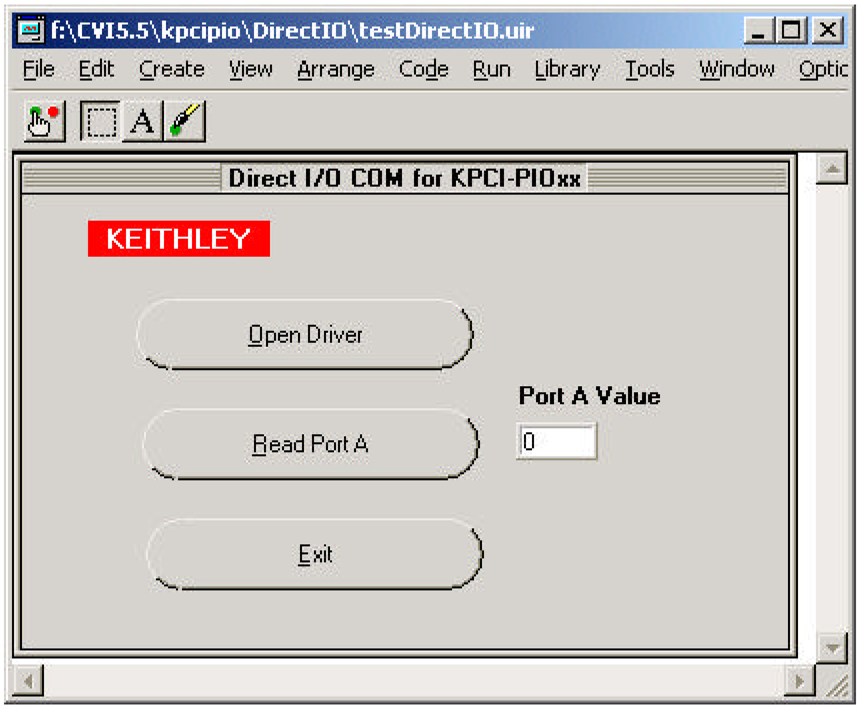
Select Function Panel

Instrument: Keithley KPCI-PIO Direct I/O Library
Class: IKPCIPIO

Window or Class

New IKPCIPIO
Open IKPCIPIO
Active IKPCIPIO
OpenDevice
CloseDevice
Read
Write
GetOpenCount
GetDevice

Step 7: Create a project called testDirectIO. Add the KPCIDIO.fp file to the project. Add also a uir file to the project. Place several buttons onto the user interface as shown below and create all the callback code stubs (a *.c file is created).

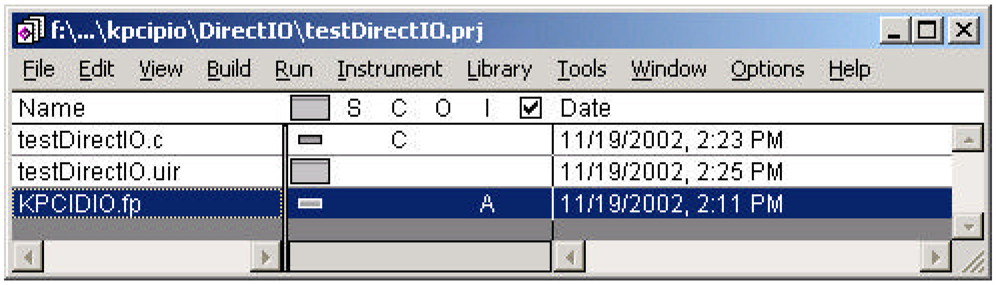

f:\CVI5.5\kpcipio\DirectIO\testDirectIO.uir
File Edit Create View Arrange Code Run Library Tools Window Optic
Direct I/O COM for KPCI-PIOxx
KEITHLEY
Open Driver
Read Port A
Exit
Port A Value
0

In the call back for the Open Driver button, the Direct IO API will be initialized and a session opened for the installed board (the device number used in the DriverLINX Configuration Panel will be used).

In the Read Port A button, the register corresponding to Port A will be read to obtain the logic 0 or 1 state of the 8-line channel. A value between 0 and 255 will result.

In the Exit button, the driver will be closed.

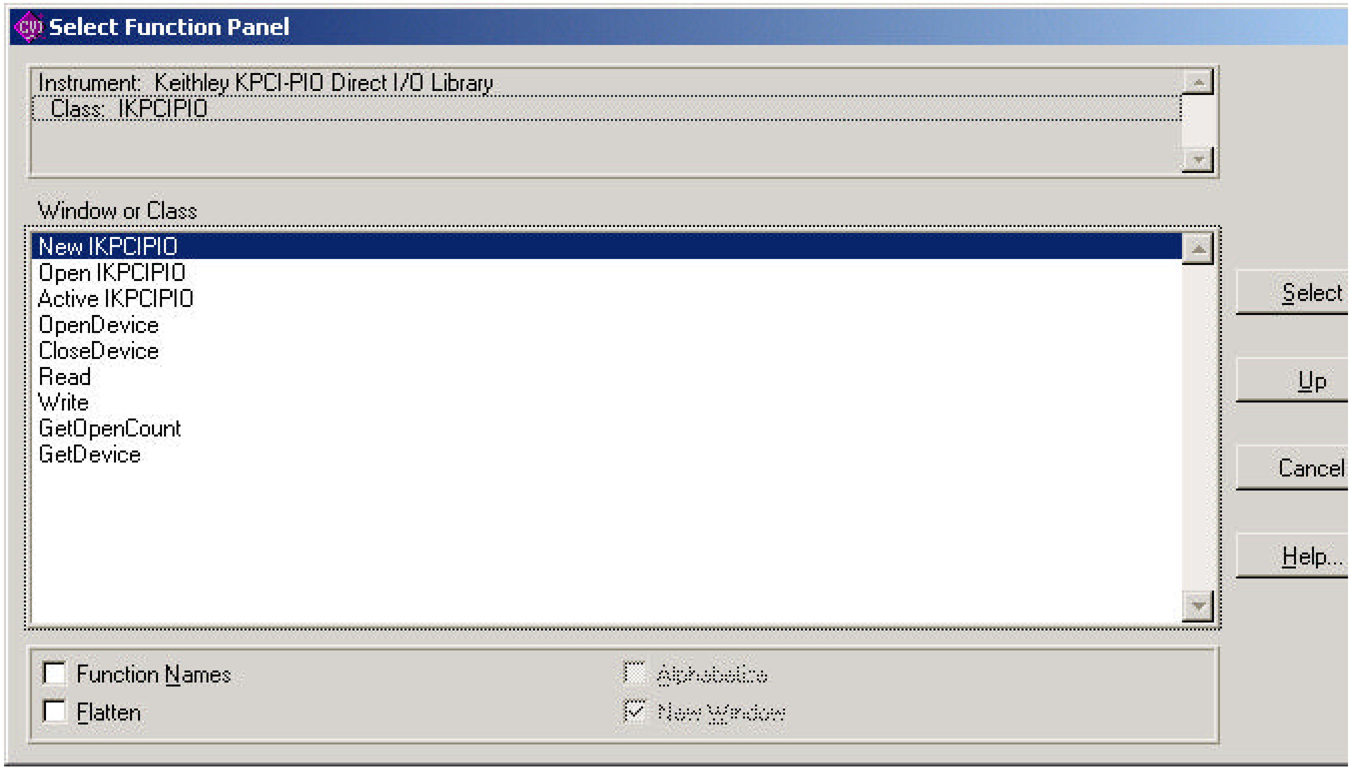
Step 8: Here is what the project will look like with it's three files.


f:\...\kpcipio\DirectIO\testDirectIO.prj

File Edit View Build Run Instrument Library Tools Window Options Help

Name		S	C	O	I	☑	Date
testDirectIO.c	▬		C				11/19/2002, 2:23 PM
testDirectIO.uir	▭						11/19/2002, 2:25 PM
KPCIDIO.fp	▬				A		11/19/2002, 2:11 PM

Step 9: Let's use the fp file to add code to our .c file. Select the NewIKPCIPIO function of the IKPCIPIO class (not the IHardware class) and then click the Select button on the right.

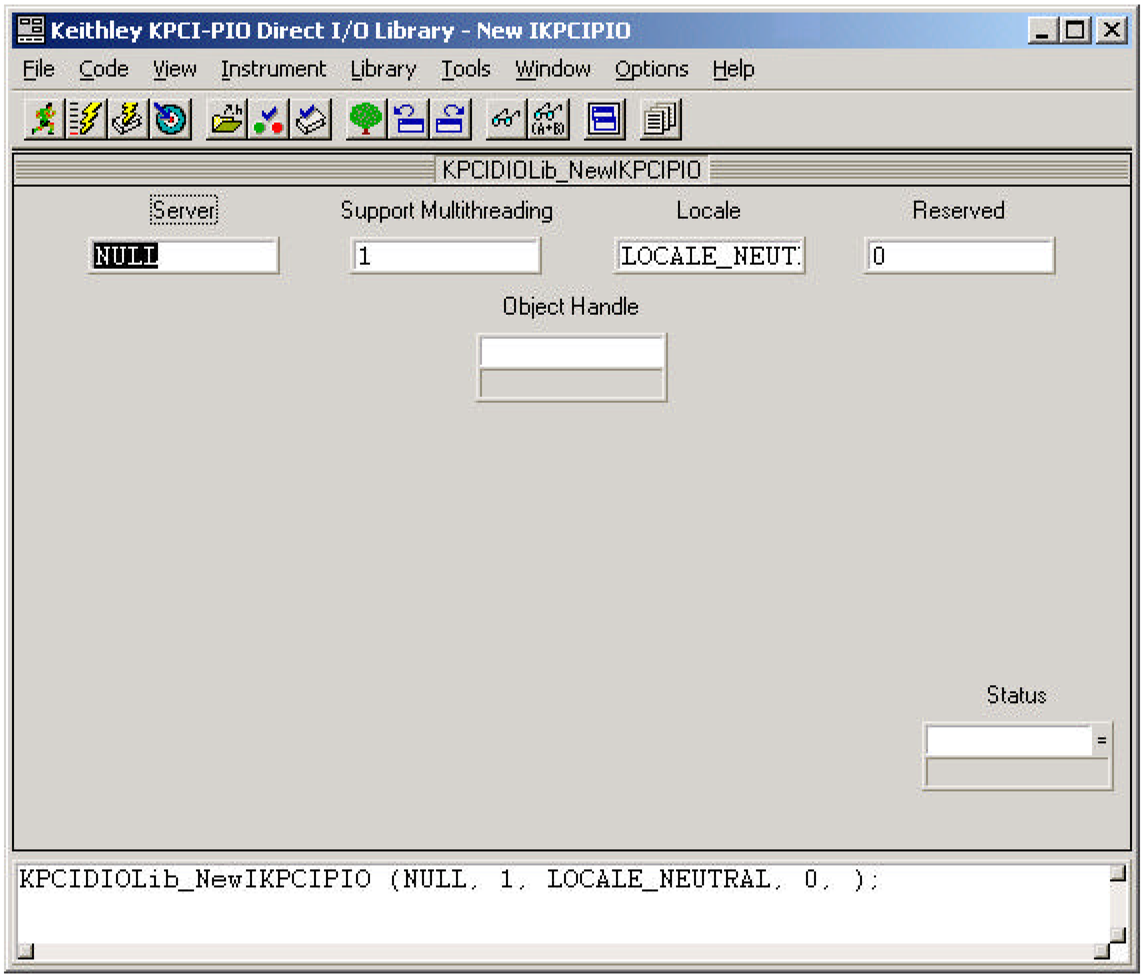

Select Function Panel

Instrument: Keithley KPCI-PIO Direct I/O Library
Class: IKPCIPIO

Window or Class

New IKPCIPIO
Open IKPCIPIO
Active IKPCIPIO
OpenDevice
CloseDevice
Read
Write
GetOpenCount
GetDevice

Step 10: the screen below is presented to aid in the use of the function. Notice the variable Object Handle is not filled in with a value. This indicates it needs to be declared by you. Select this object with the mouse, and then ask CVI to declare the variable (Ctrl+D, or menu structure of Code or the short cut buttons). (The Status variable could also be declared for a more robust implementation that includes error checking, but for brevity, this is skipped in this application note.)


Keithley KPCI-PIO Direct I/O Library - New IKPCIPIO
File Code View Instrument Library Tools Window Options Help
KPCIDIOLib_NewIKPCIPIO

Server: NULL

Support Multithreading: 1

Locale: LOCALE_NEUT

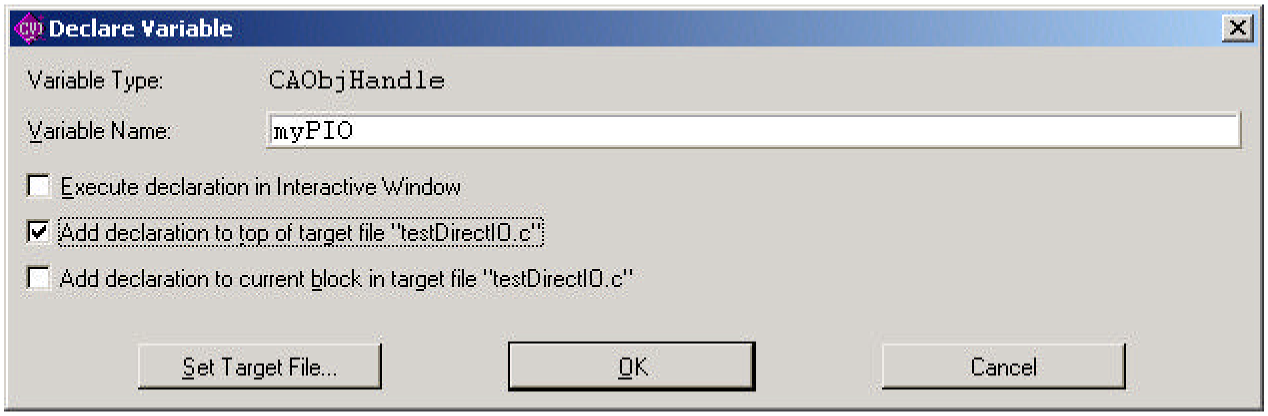
Reserved: 0

Object Handle:

Status:

```
KPCIDIOLib_NewIKPCIPIO (NULL, 1, LOCALE_NEUTRAL, 0, );
```

Step 11: The following dialog is presented for declaring our object handle variable. Fill in a variable name; I used myPIO. Also check the box indicating the declaration should be added to the top of the target .c file for our project. Then click the OK button.



Variable Type: CAObjHandle

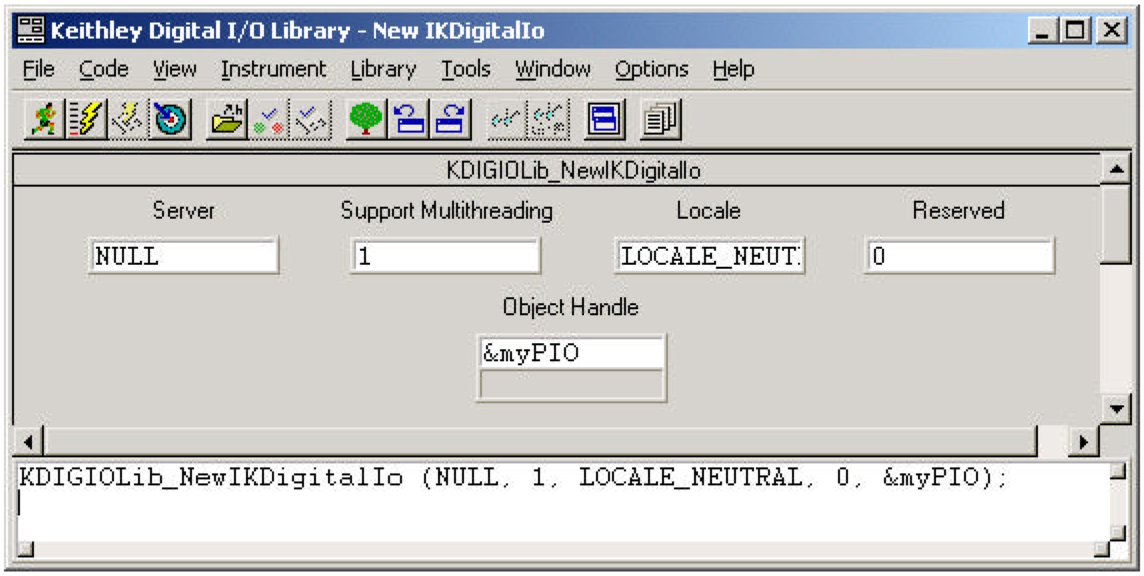
Variable Name: myPIO

☐ Execute declaration in Interactive Window

☑ Add declaration to top of target file "testDirectIO.c"

☐ Add declaration to current block in target file "testDirectIO.c"

Step 12: Notice the field is now filled in with the proper calling syntax (address of operator). In the .c file, verify the curser is within the callback function for the Open button, and then back on the screen below ask CVI to "insert function call" using the Code menu structure.

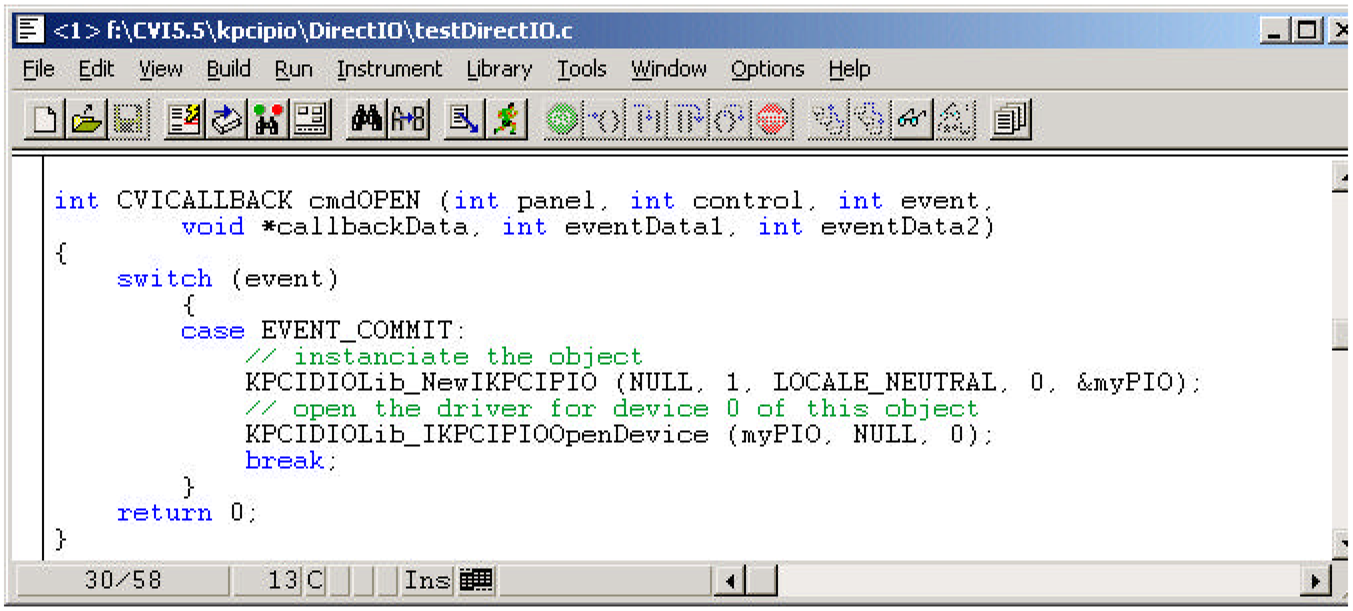


Server	Support Multithreading	Locale	Reserved
NULL	1	LOCALE_NEUT	0

Object Handle: &myPIO

```
KDIGIOLib_NewIKDigitalIo (NULL, 1, LOCALE_NEUTRAL, 0, &myPIO);
```

Step 13: Below is a screen shot of the callback for the open button in the .c file after two functions of the Direct IO COM object have been used. All subsequent methods of the Direct I/O object require the Object Handle obtained from the NewIKPCIPIO method. Notice the syntax used with the OpenDevice method (no & operator).



<1> f:\CVI5.5\kpcipio\DirectIO\testDirectIO.c
File Edit View Build Run Instrument Library Tools Window Options Help
30/58 13 C Ins

```
int CVICALLBACK cmdOPEN (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
        {
        case EVENT_COMMIT:
            // instanciate the object
            KPCIDIOLib_NewIKPCIPIO (NULL, 1, LOCALE_NEUTRAL, 0, &myPIO);
            // open the driver for device 0 of this object
            KPCIDIOLib_IKPCIPIOOpenDevice (myPIO, NULL, 0);
            break;
        }
    return 0;
}
```

Once the object has been created and the driver opened for the installed device, the Read and Write functions can be used to get data into and out of the KPCI card. You will need to know something of the 8255 architecture to know at what offsets (address) and what values to write. Below is a brief summary:

Port A is at an offset of 0
Port B is at an offset of 1
Port C is at an offset of 2

The control register is at an offset of 3. Individual bits of this register control the input or output direction of the individual ports. Write a 0 to the bit to make the corresponding Port an output; write a 1 to make it an input. All ports are inputs by default at PC power-up or reset. If using only digital inputs, then the control register is not important for your application.

bit 0.......4 lower bits of Port C
bit 1.......Port B
bit 2.......not used, always zero
bit 3.......4 upper bits of Port C
bit 4.......Port A
bit 5.......not used, always zero
bit 6.......not used, always zero
bit 7.......not used, **always 1**

To make all ports outputs, write a hex value of 80 to the control register and then write to the registers for the individual ports to control their logic 0 or 1 state.

Consult chapter 7 of the DriverLINX Tutorial Manual or the Using DriverLINX with your KPCI-PIO Series manual for more information.

A complete CVI project is available in the download center of the www.keithley.com web site.