Java I/O Driver

API Specification
(Version 0.13b)

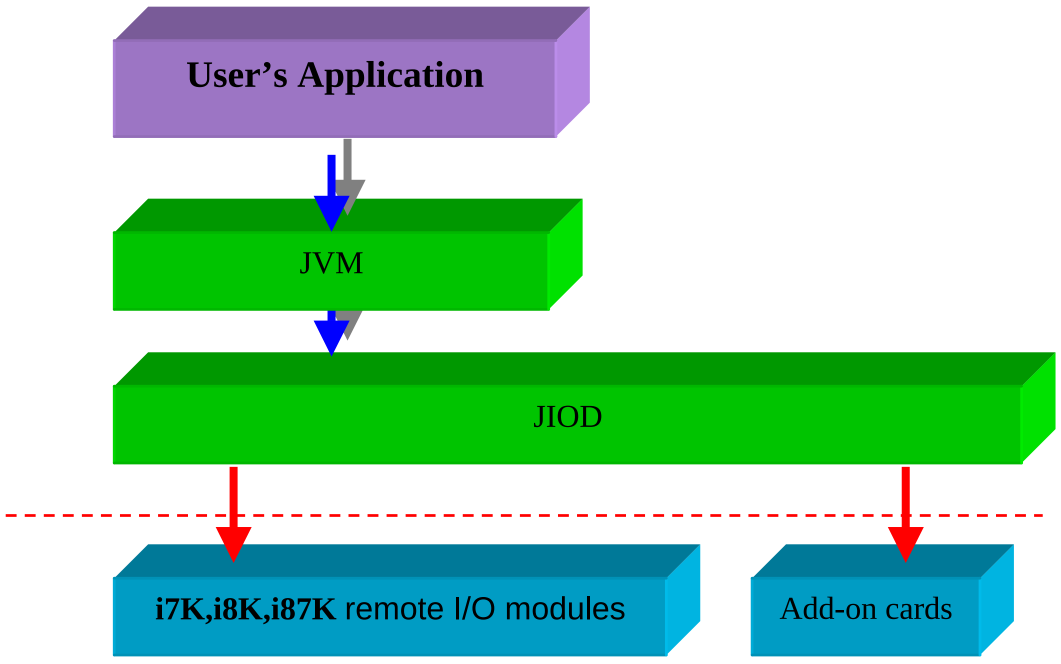
Table of Contents

1. Introduction

Java I/O Driver (JIOD) are the Java platform technology of choice for extending and enhancing JVM to made many industry control applications possible. JIOD included I/O packages for i7k,i8k,i87k remote I/O modules and PCI bus series add-on cards. JIOD provides developers with a simple and easy mechanism for extending the functionality of JVM and for accessing ICPDAS products.

The JIOD contain three packages com.icpdas.ixpio,com.icpdas.ixpci and com.icpdas.comm. Ixpio and ixpci packages support PCI bus series add-on cards and comm packages support i7k,i8k,i87k remote I/O modules.

The packages in JIOD are easy understanding as its name standing for. It provides powerful, easy-to-use packages for developing your data acquisition application. Program can use these packages within application, applet and servlet easily. To speed-up your developing process, some demonstration source program are provided. The relation between JIOD and user's application depicted as follows:



(JIOD ARCHITECTURES)

2. USING JIOD

The usage of JIOD is very similar to that for C user. The key points are given as following:

- Add icpdas.jar to CLASSPATH.
- Import JIOD in source program.

Examples:

```
import com.icpdas.cardio.ixpio.*;              //For System.in.read()
import com.icpdas.comm.*;              //ICPDAS communication packages

public class Dio
{
    public static void main(String[] args)
    {
      int rev;
      int fd;
      Comm comm1 = new Comm();           //ICPDAS communication object
      IoBuf i7kBuf = new IoBuf();           //control matrix
      rev = comm1.open(1,9600,comm1.DATABITS_8,comm1.PARITY_NONE,comm1.STOPBITS_1);
        //open serial port
      if (rev!=0) System.out.println("Open port error code : "+rev);
      else{
              i7kBuf.dwBuf[0] =1;                   //Serial Port no
              i7kBuf.dwBuf[1] =1;                   //Address
              i7kBuf.dwBuf[2] =0x7060;           //0x7060; //module name
              i7kBuf.dwBuf[3] =0;                   //check sum disable
              i7kBuf.dwBuf[4] =10;                  //Timeout 100ms
              rev = comm1.getDigitalIn(i7kBuf);              //Get Digital Input Value from 7060
              if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
              else System.out.println("Digital In : "+ i7kBuf.dwBuf[5]);
      }
      comm1.close(1);
      System.out.println(rev);
      System.out.println("End of program");
    }
}
```

3. API Documentation

The JIOD contain three packages to support variant ICPDAS product. Before use specified device must import correspond package first.

The device that JIOD have support:
(1)PIO Series I/O cards
(2)PCI Series I/O cards
(3)i7k,i8k and i7k Series Modules

Packages Summary	
com.icpdas.comm	For i7k,i87,i87k series remote I/O modules
com.icpdas.ixpci	For PCI series add-on cards
com.icpdas.ixpio	For PIO series add-on cards

3.1 com.icpdas.comm Package

Package com.icpdas.comm Description:
The com.icpdas.comm package can be used to write platform-independent industry applications. Provides the classes necessary to control remote modules. ICPDAS remote I/O modules provide cost-effective protection and conditioning for a wide range of valuable industrial control signals and system. The command set of modules is backward compatible to ADAM, Nudam, and 6B of Analog Device.

(ICPDAS remote I/O modules)

Class Summary	
Comm	Defines methods that communication to serial device
IoBuf	Remote modules control matrix.

3.1.1 Class Comm

com.icpdas.comm

Class Comm

```
java.lang.Object
  |
  +--com.icpdas.comm.Comm
```

The Comm Class included both low level serial communication method and high level remote I/O modules control method. A serial port can be opend for reading and writing data. Once the application is done with the port, it must call the close method before end program.

Method Summary	
int	open(int portno, int baudrate, int databit, int isparity, int stopbit) Initialize the COM port.
void	close(int portno) Free all the resources used by open. This method must be **called before** the program exit.
int	setSendCmd(IoBuf ioArg) Create a thread to send a command to module.
int	getReceiveCmd(IoBuf ioArg) Create a thread to receive the response-result from module.
int	getSendReceiveCmd(IoBuf ioArg) Create a thread to send a command and receive the response-result from module.
int	getAnalogIn(IoBuf ioArg) Read the analog input value from module.
int	getAnalogInAll(IoBuf ioArg) Read all channels of analog input values from module.
int	setAnalogOut(IoBuf ioArg) Send the analog output command to module.
int	getAnalogOutReadBack(IoBuf ioArg) Read back the current D/A output value of module.
int	getDigitalIn(IoBuf ioArg) Read the digital input value from module.
int	setDigitalOut(IoBuf ioArg) To set the digital output value for module.
int	getDigitalOutReadBack(IoBuf ioArg) Read back the digital output value of module.

int	setDigitalBitOut(IoBuf ioArg) Set the digital value of digital output channel No.
int	getDigitalInLatch(IoBuf ioArg) Obtain the latch value of the high or low latch mode of Digital Input module.
int	setClearDigitalInLatch(IoBuf ioArg) Clear the latch status of digital input module when latch function has been enabled.
int	getDigitalInCounter(IoBuf ioArg) Obtain the counter event value of the channel number of Digital Input module.
int	setClearDigitalInCounter(IoBuf ioArg) Clear the counter value of the digital input channel No.
int	setAlarmMode (IoBuf ioArg) To set module enter *momentary alarm mode* or *latch alarm mode*.
int	setAlarmConnect(IoBuf ioArg) Set the link between DO and AI module.
int	setClearLatchAlarm(IoBuf ioArg) To clear the latch alarm for module.
int	setAlarmLimitValue(IoBuf ioArg) To set a high or low alarm limit value for module.
int	getAlarmLimitValue(IoBuf ioArg) To get the high or low alarm limit value for module.
int	getAlarmStatus(IoBuf ioArg) Reading the alarm status for a module.
int	getAlarmMode(IoBuf ioArg) Reading the alarm mode for a module.
int	getConfigStatus(IoBuf ioArg) Obtain the configuration status of the modules.
int	setStartUpValue(IoBuf ioArg) Configure the initial analog output of analog output module when its power is on.
int	getStartUpValue(IoBuf ioArg) Obtain the initial output setting value of analog output module when the power is on.

Field Summary

DATABITS_5, **DATABITS_6**, **DATABITS_7**, **DATABITS_8**, **PARITY_EVEN**, **PARITY_NONE**, **PARITY_ODD**, **STOPBITS_1**, **STOPBITS_1_5**, **STOPBITS_2**

Method Detail

open
public int **open**(int portno, int baudrate, int databit, int isparity, int stopbit)

Initialize the COM port. This method must be **called once before** the other method are called to send/receive command.

Example:
```
import com.icpdas.cardio.ixpio.*;                    //For System.in.read()
import com.icpdas.comm.*;                            //ICPDAS communication packages

public class Dio
{
    public static void main(String[] args)
    {
        int rev;
        int fd;
        Comm comm1 = new Comm();                    //ICPDAS communication object
        IoBuf i7kBuf = new IoBuf();                 //control matrix
        rev =
comm1.open(1,9600,comm1.DATABITS_8,comm1.PARITY_NONE,comm1.STOPBITS_1);//open
serial port
        if (rev!=0) System.out.println("Open port error code : "+rev);
        else{
            i7kBuf.dwBuf[0] =1;                     //Serial Port no
            i7kBuf.dwBuf[1] =1;                     //Address
            i7kBuf.dwBuf[2] =0x7060;                //0x7060; //module name
            i7kBuf.dwBuf[3] =0;                     //check sum disable
            i7kBuf.dwBuf[4] =10;                    //Timeout 100ms
            rev = comm1.getDigitalIn(i7kBuf);       //Get Digital Input Value from 7060
            if (rev!=0) System.out.println("Digital In Error Code : "+ rev);
            else System.out.println("Digital In : "+ i7kBuf.dwBuf[5]);
        }
        comm1.close(1);
        System.out.println(rev);
        System.out.println("End of program");
    }
}
```

See Also:close, getDigitalIn

close
public void **close**(int portno)

Free all the resources used by open. This method must be **called before** the program exit. The open will return error message if the program exit without calling close method.

See Also:open

setSendCmd
public int **setSendCmd**(IoBuf ioArg)

This method will create a thread to send a command to module. If the wCheckSum=1, this method will automatically **add the two checksum bytes** to the input string. This method will **add the [0x0D]** to the end of the input string, szCmd.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.szSend	command string send to module

Example:

```
import com.icpdas.comm.*;                    //ICPDAS communication packages

public class Send
{
   public static void main(String[] args)
   {
      int rev;
      int fd,i=1;
      byte a[] = new byte[100];
      Comm comm1 = new Comm();            //ICPDAS communication object
      IoBuf i7kBuf = new IoBuf();             //control matrix
      rev =
comm1.open(1,9600,comm1.DATABITS_8,comm1.PARITY_NONE,comm1.STOPBITS_1);//open
serial port
      if (rev!=0) System.out.println("Open port error code : "+rev);
      else{
         i7kBuf.dwBuf[0] = 1;               //Serial Port no
         i7kBuf.dwBuf[3] = 0;               //check sum disable
         i7kBuf.dwBuf[4] = 10;              //Timeout 100ms
         i7kBuf.szSend = "icpdasICONportocaltest"+i;
         rev = comm1.setSendCmd(i7kBuf);          //Send command to i7060 module
         rev = comm1.getReceiveCmd (i7kBuf);          //Send command to i7060 module
      }
      System.out.println("Receive = "+ i7kBuf. szReceive);
      comm1.close(1);                     //close serial port
      System.out.println("End of program");
   }
}
```

See Also:open,close, getReceiveCmd, getSendReceiveCmd

getReceiveCmd
public int **getReceiveCmd**(IoBuf ioArg)

This method will create a thread to receive the response-result from module. If the wCheckSum=1, this method will automatically **check the two checksum bytes** in the receive string.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[3]	checksum enable or disable

IoBuf.dwBuf[4]	timeout value

Return Value

IoBuf.szReceive	result string read from module

See Also:open,close, setSendCmd, getSendReceiveCmd

---

getSendReceiveCmd
public int **getSendReceiveCmd**(IoBuf ioArg)

This method will create a thread to send a command and receive the response-result from module. If the wCheckSum=1, this method will automatically **add the two checksum bytes** to the input string. This method will **add the [0x0D]** to the end of the input string, szCmd.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value

Return Value

IoBuf.szReceive	result string read from module

See Also:open,close, setSendCmd, getReceiveCmd

---

getAnalogIn
public int **getAnalogIn**(IoBuf ioArg)

Read the analog input value from analog module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	analog channel number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.fBuf[0]	analog input value return

Example:

```
import java.io.*;
import com.icpdas.comm.*;

public class Aio
{
   public static void main(String[] args) throws java.io.IOException
   {
      int rev;
      int fd,ao=1;
      byte a[] = new byte[100];
      Comm comm1 = new Comm();
      IoBuf i7kBuf = new IoBuf();
      rev =
comm1.open(1,9600,comm1.DATABITS_8,comm1.PARITY_NONE,comm1.STOPBITS_1);
      if (rev!=0) System.out.println("Open port error code : "+rev);
      else{
         i7kBuf.dwBuf[0] = 1;      //Serial Port
         i7kBuf.dwBuf[1] = 3;      //Address
         i7kBuf.dwBuf[3] = 0;      //check sum disable
         i7kBuf.dwBuf[4] = 10;     //Timeout 100ms
         i7kBuf.dwBuf[6] = 1;      //Enable String Debug
         while(a[0]!=113) {
            i7kBuf.dwBuf[2] = 0x7016 ;//0x7016; //module name
            i7kBuf.fBuf[0] = ao;
            rev = comm1.setAnalogOut(i7kBuf);
            if (rev!=0) System.out.println("Analog Out Error Code : "+ rev);
            System.out.println("szSend = "+ i7kBuf.szSend +" szReceive = "+i7kBuf.szReceive);
            i7kBuf.dwBuf[2] = 0x7012 ;//0x7012; //module name
            rev = comm1.getAnalogIn(i7kBuf);
            if (rev!=0) System.out.println("Analog In Error Code : "+ rev);
            System.out.println("szSend = "+ i7kBuf.szSend +" szReceive = "+i7kBuf.szReceive);
            System.out.println("Analog In Value : "+i7kBuf.fBuf[0]);
            System.in.read(a);
            ao=(ao>128)?1:(ao<<1);
         }
      }
      comm1.close(1);
      System.out.println("End of program");
   }
}
```

See Also:open,close

getAnalogInAll

public int **getAnalogInAll**(IoBuf ioArg)

Read the all channels of analog input values from analog module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	analog channel number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive

	1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.fBuf[0]	analog input channel 0 value return
IoBuf.fBuf[1]	analog input channel 1 value return
IoBuf.fBuf[2]	analog input channel 2 value return

•
•
•

See Also:open,close

setAnalogOut

public int **setAnalogOut**(IoBuf ioArg)

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	analog channel number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only
IoBuf.fBuf[0]	analog output value

See Also:open,close

getAnalogOutReadBack

public int **getAnalogOutReadBack**(IoBuf ioArg)

Read back the current D/A output value of module. There are two types of analog output read back described as following:

1. **command read back by $AA6 command**
2. **analog output of current path read back by $AA8 command**

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable

IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module

for i7k,i87k series

IoBuf.dwBuf[5]	readback type	0 → command read back ($AA6) 1 → current path read back ($AA8)
IoBuf.dwBuf[7]	channel number for multi-channel	

for i8k series

IoBuf.dwBuf[5]	channel number for multi-channel
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.fBuf[0]	analog output value readback

See Also:open,close

getDigitalIn

public int **getDigitalIn**(IoBuf ioArg)

Read the digital input value from a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	

Return Value

IoBuf.dwBuf[5]	16-bit digital input data

See Also:open,close

setDigitalOut

public int **setDigitalOut**(IoBuf ioArg)

To set the digital output value for a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	16-bit digital output data
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

See Also:open,close

getDigitalOutReadback

public int **getDigitalOutReadback** (IoBuf ioArg)

Readback the digital out value from a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.dwBuf[5]	16-bit digital readback data

See Also:open,close

setDigitalBitOut

public int **setDigitalBitOut**(IoBuf ioArg)

Set the digital value of digital output channel No.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value

IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module

for i7k,i87k series

IoBuf.dwBuf[7]	which channel to output	
IoBuf.dwBuf[8]	output data	0 → output low 1 → output high

for i8k series

IoBuf.dwBuf[5]	output data	0 → output low 1 → output high
IoBuf.dwBuf[7]	Slot Number for i8k series only	
IoBuf.dwBuf[8]	which channel to output	

See Also:open,close

---

getDigitalInLatch
public int **getDigitalInLatch**(IoBuf ioArg)

Obtain the latch value of the high or low latch mode of Digital Input module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	output data	0 → Latch Low 1 → Latch High
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	

Return Value

IoBuf.dwBuf[5]	latch value

See Also:open,close

---

setClearDigitalInLatch
public int **setClearDigitalInLatch**(IoBuf ioArg)

Clear the latch status of digital input module when latch function has been enabled.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

See Also:open,close

getDigitalInCounter
public int **getDigitalInCounter**(IoBuf ioArg)

Obtain the counter event value of the channel number of Digital Input module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.dwBuf[7]	Counter Value of Channel N's Digital Input

See Also:open,close

setClearDigitalInCounter
public int **setClearDigitalInCounter**(IoBuf ioArg)

Clear the counter value of the digital input channel No.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number

IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	

See Also:open,close

---

setAlarmMode

public int **setAlarmMode** (IoBuf ioArg)

To set module enter momentary alarm mode or latch alarm mode.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	Channel Number	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	
IoBuf.dwBuf[8]	Alarm mode	0 → Low Alarm 1 → High Alarm
IoBuf.dwBuf[9]	Alarm State	0 → Disable 1 → Momentary 2 → Latch

See Also:open,close

---

setAlarmConnect

public int **setAlarmConnect**(IoBuf ioArg)

8000 Series provide a method to output via DO module when certain module alarm. Use SetAlarmConnect_8K function to set the link between DO and AI module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	Channel Number	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module

IoBuf.dwBuf[7]	Slot Number for i8k series only
IoBuf.dwBuf[8]	Alarm mode 0 → Low Alarm 1 → High Alarm
IoBuf.dwBuf[9]	The slot that desired connect DO module in.
IoBuf.dwBuf[10]	The channel that desired connect DO channel.

See Also:open,close

setClearLatchAlarm

public int **setClearLatchAlarm**(IoBuf ioArg)

To clear the latch alarm for a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only
IoBuf.dwBuf[8]	Alarm mode 0 → Low Alarm 1 → High Alarm

See Also:open,close

setAlarmLimitValue

public int **setAlarmLimitValue**(IoBuf ioArg)

To set a high or low alarm limit value for module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

IoBuf.dwBuf[8]	Alarm mode	0 → low alarm value setting 1 → high alarm value setting
IoBuf.fBuf[0]	Alarm value	

See Also:open,close

---

getAlarmLimitValue

public int **getAlarmLimitValue**(IoBuf ioArg)

To get the high or low alarm limit value for module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	Channel Number	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	
IoBuf.dwBuf[8]	Alarm mode	0 → low alarm value setting 1 → igh alarm value setting

Return Value

IoBuf.fBuf[0]	Alarm value

See Also:open,close

---

getAlarmStatus

public int **getAlarmStatus**(IoBuf ioArg)

Reading the alarm status for a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	Channel Number	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	

Return Value

IoBuf.dwBuf[8]	Alarm status	0 → High Alarm Don't Occur
		1 → High Alarm Occur
IoBuf.dwBuf[9]	Alarm status	0 → Low Alarm Don't Occur
		1 → Low Alarm Occur

See Also:open,close

---

getAlarmMode

public int **getAlarmMode**(IoBuf ioArg)

Reading the alarm mode for a module.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.	
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF	
IoBuf.dwBuf[2]	module ID	
IoBuf.dwBuf[3]	checksum enable or disable	
IoBuf.dwBuf[4]	timeout value	
IoBuf.dwBuf[5]	Channel Number	
IoBuf.dwBuf[6]	debug string	0 → no save to szSend&szReceive
		1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only	
IoBuf.dwBuf[8]	Alarm mode	0 → Low Alarm
		1 → High Alarm

Return Value

IoBuf.dwBuf[9]	Alarm State	0 → Disable
		1 → Momentary
		2 → Latch

See Also:open,close

---

getConfigStatus

public int **getConfigStatus**(IoBuf ioArg)

Obtain the configuration status of the modules.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number

IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value
for i7k,i87k series

IoBuf.dwBuf[7]	Module address
IoBuf.dwBuf[8]	Module Range Code
IoBuf.dwBuf[9]	Module Baudrate
IoBuf.dwBuf[10]	Module data format

for i8k series

IoBuf.dwBuf[8]	Output Range
IoBuf.dwBuf[9]	Slew rate

See Also:open,close

setStartUpValue
public int **setStartUpValue**(IoBuf ioArg)

Configure the initial analog output of analog output module when its power is on.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID
IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only
IoBuf.fBuf[0]	Start-Up Value

See Also:open,close

getStartUpValue
public int **getStartUpValue**(IoBuf ioArg)

Obtain the initial output setting value of analog output module when the power is on.

Input Parameter

IoBuf.dwBuf[0]	Serial Port No.
IoBuf.dwBuf[1]	module address, from 0x00 to 0xFF
IoBuf.dwBuf[2]	module ID

IoBuf.dwBuf[3]	checksum enable or disable
IoBuf.dwBuf[4]	timeout value
IoBuf.dwBuf[5]	Channel Number
IoBuf.dwBuf[6]	debug string 0 → no save to szSend&szReceive 1 → szSend= command string send to module szReceive= string receive from module
IoBuf.dwBuf[7]	Slot Number for i8k series only

Return Value

IoBuf.fBuf[0]	Start-Up Value

See Also:open,close

Field Detail

static int	**DATABITS_5** 5 data bit format.
static int	**DATABITS_6** 6 data bit format.
static int	**DATABITS_7** 7 data bit format.
static int	**DATABITS_8** 8 data bit format.
static int	**PARITY_EVEN** EVEN parity check.
static int	**PARITY_NONE** No parity bit.
static int	**PARITY_ODD** ODD parity check.
static int	**STOPBITS_1** Number of stop bits - 1.
static int	**STOPBITS_1_5** Number of stop bits - 1-1/2.
static int	**STOPBITS_2** Number of stop bits - 2.

3.1.2 Class IoBuf

com.icpdas.comm

Class IoBuf

```
java.lang.Object
  |
  +--com.icpdas.comm.IoBuf
```

IoBuf class provide variable that high level remote I/O modules control method use.

Field Summary

dwBuf, **fBuf**, **szSend**, **szReceive**

Field Detail

dwBuf
public int **dwBuf[]**
Double word length matrix for module control.

fBuf
public float **fBuf[]**
Floating matrix for module control.

szSend
public java.lang.String **szSend**
Command string send to module.

szReceive
public java.lang.String **szSend**
String receive from module.

3.2 com.icpdas.ixpci Package

Package com.icpdas.ixpci Description:
The com.icpdas.ixpci package can be used to write platform-independent industry applications. Provides the classes necessary to support pci series add-on cards. ICPDAS pci add-on cards series is a family of high performance data acquisition board for PC with PCI bus.

(ICPDAS PCI-1800 add-on cards)

Interface Summary	
IxpciIsr	Provide a interface to create a routine to handle a interrupt event.

Class Summary	
Ixpci	Defines methods that communication to pci series add-on cards.
IxpciDio	Digital input/output variable.
IxpciAio	Analog input/output variable.
IxpciReg	Register access variable.
IxpciInfo	Add-on cards peripheral information variable.

3.2.1 Class IxpciIsr

com.icpdas.cardio.ixpci

Class IxpciIsr

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpci.IxpciIsr
```

Interrupt Service Routine (ISR) was a small routine that execute when interrupt happen. The IxpciIsr class Provide a interface to create a routine to handle a interrupt event.

Method Summary	
void	ixpci_isr(int sig) Called by JIOD to allow a routine to handle a interrupt request.

Method Detail

ixpci_isr
public void **ixpci_isr**(int sig)

The way to create a ISR is to declare a class that implements the IxpciIsr interface. That class then implements the ixpci_isr method. When the interrupt event occurs, that IxpciIsr's ixpci_isr method is invoked.

Example:

```
import java.io.*;                          //For System.in.read()
import com.icpdas.cardio.ixpci.*;          //ICPDAS PCI Series I/O Card packages


public class Interrupt implements IxpciIsr     //Implement IxpioIsr interrupt class
{
   static int beat = 0;
   static int INT1 = 2;
   static int INT1_NEG_EDGE = 0;
   static IxpciDio din,dout;
   static Ixpci icpdasio1;
   static int fd;
   public void ixpci_isr(int sig)              //Will be call when trigger interrupt
   {
      int rev;
      if ((rev=icpdasio1.setDigitalOut(fd,dout))!=0) System.out.println("DO Error");
      if (dout.u16 == 0x8000) dout.u16 = 1;
```

```
        else dout.u16 <<=1 ;
        beat ++;
    }

    public static void main(String[] args) throws java.io.IOException
    {
        int rev;
        Interrupt demo_isr;
        byte a[] = new byte[100];
        icpdasio1=new Ixpci();
        din = new IxpciDio();
        dout = new IxpciDio();
        fd=icpdasio1.open("ixpci1");
        if(fd==-1) System.out.println("Card Open Error Code:"+ fd);
        din.u16 = 0;
        dout.u16 = 1;
        demo_isr = new Interrupt();
        /* INT_CHAN_1, signal for negative edges*/
        rev=icpdasio1.registerInterrupt(fd, INT1, INT1_NEG_EDGE, demo_isr);
        if(rev != 0) System.out.println("Install Interrupt Error Code:"+ rev);
        // counter 2 on chip 3, counter 12 on board, mode 3
        rev=icpdasio1.setCounter(fd,11,0,0xb6,0x9c3f);
        if(rev != 0) System.out.println("Config Counter 1 Error Code:"+ rev);
        while(a[0]!=10)
        {
            System.in.read(a);
        }
        /*Remove Interrupt before program end*/
        rev=icpdasio1.removeInterrupt(fd);
        if(rev != 0) System.out.println("Remove Interrupt Error Code:"+ rev);
        icpdasio1.close(fd);
        System.out.println("End of program");
    }
}
```

See Also:open,close

3.2.2 Class Ixpci

com.icpdas.cardio.ixpci

Class Ixpci

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpci.Ixpci
```

The Ixpci Class provide pci series add-on cards control method. A pci series add-on cards can be opend for reading and writing data. Once the application is done with the port, it must call the close method before end program.

Method Summary

int	open(java.lang.String filename) Initialize the pci card.
void	close(int cardfd) Free all the resources used by open. This method must be **called before** the program exit.
int	setPort(int cardfd, java.lang.String portid, int portattr) Configure I/O port attribute.
int	setReg(int cardfd, java.lang.String regname, IxpciReg IxpciArg) Write value to card register.
int	getReg(int cardfd, java.lang.String regname) Obtion register value of card.
int	getInfo(int cardfd, IxpciInfo IxpciArg) Obtion peripheral information of a card.
int	getDigitalIn(int cardfd, IxpciDio IxpciArg) Read the digital input value from card.
int	setDigitalOut(int cardfd, IxpciDio IxpciArg) To set the digital output value for card.
int	getAnalogIn(int cardfd, IxpciAio IxpciArg) Read the analog input value from card.
int	setAnalogOut(int cardfd, IxpciAio IxpciArg) Send the analog output command to card.
int	registerInterrupt(int cardfd,int interruptchannel,int eddgetype,IxpciIsr javaisr) Install user define interrupt service routine.
int	removeInterrupt(int cardfd) Remove user define interrupt service routine.
int	setCounter(int cardfd,int counterno,int cicr,int mode,int value) Configure counter on add-on card.

Method Detail

open
public int **open**(java.lang.String filename)

This method will open the card and allocate the resource for the device. This function must be called once before calling other method.

Input Parameter

filename	/dev/ixpcin	The n is the PCI device number.

Return Value

fd	File descriptor number of pci card.

See Also close

---

close

public void **close**(int cardfd)

This method will close the pci card and release the resource from the device. This method must be called once before exit the user's application.

Input Parameter

fd	File descriptor no of pci card that return from open.

See Also:open

---

setPort

public int **setPort**(int cardfd, java.lang.String portname, int portattr)

Configure specified I/O port attribute.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
portname	Name of the port control register.
portattr	The port setting value.

Return Value

0	No error.

See Also:open,close

---

setReg

public int **setReg**(int cardfd, java.lang.String regname, IxpciReg IxpciArg)

Write a given value to a specified register.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
regname	Name of the register.

Return Value

0	No error.

See Also:open,close

---

getReg

public int **getReg**(int cardfd, java.lang.String regname)

Read a specified register value.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
regname	Name of the register.

Return Value

int	The register value.

See Also:open,close

---

getInfo

public int **getInfo**(int cardfd, IxpciInfo IxpciArg)

Get the I/O address of specified pci board.

Input Parameter

cardfd	File descriptor no of pci card that return from open.

Return Value

IxpciArg	Peripheral information of specified pci board.

Example:

```
import com.icpdas.cardio.ixpci.*;

public class List
{
   public static void main(String[] args)
   {
      int rev;
      int fd=0;
      int i=1;
      String devname = new String("ixpci");
      Ixpci icpdasio1=new Ixpci();
      IxpciInfo cardinfo = new IxpciInfo();
      while (fd!=-1)
      {
         fd=icpdasio1.open(devname + i);
         if (fd!=-1) //System.out.println("Open Card Error code: "+ fd);
         {
            rev=icpdasio1.getInfo(fd,cardinfo);
            if(rev==-1) System.out.println("Read Card Info Error code:"+ rev);
            System.out.println(devname + i +" CSID = "+ Integer.toHexString(cardinfo.csid));
            icpdasio1.close(fd);
         }
         i++;
```

```
        }
        System.out.println("End of program");
    }
}
```

See Also:open,close

getDigitalIn

public int **getDigitalIn**(int cardfd, IxpciDio IxpciArg)

This method will input the 16 bit data from the desired I/O port.

Input Parameter

cardfd	File descriptor no of pci card that return from open.

Return Value

IxpciArg.u16	Digital input value.
0	No error.

See Also:open,close

setDigitalOut

public int **setDigitalOut**(int cardfd, IxpciDio IxpciArg)

This method will send the 16 bits data to the desired I/O port.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
IxpciArg.u16	Digital output value.

Return Value

0	No error.

See Also:open,close

getAnalogIn

public int **getAnalogIn**(int cardfd, IxpciAio IxpciArg)

Read the value of analog from the specified board and channel.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
IxpciArg.u16	Analog input value.

Return Value

0	No error.

See Also:open,close

setAnalogOut
public int **setAnalogOut**(int cardfd, IxpciAio IxpciArg)

Output the value of analog to the specified board and channel.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
IxpciArg.u16	Analog output value.

Return Value

0	No error.

See Also:open,close

registerInterrupt
public int **registerInterrupt**(int cardfd,int interruptchannel,int eddgetype,IxpciIsr javaisr)

This method will install the interrupt service routine. This function supports multiple interrupt-source and the Active-Mode can setting to "Active-Low only", "Active-High only" and "Active-Low or Active-High".

Caution:The ISR method must use carefully and been remove before exit program otherwise maybe cause system unstable.

Input Parameter

cardfd	File descriptor no of pci card that return from open.	
interruptchannel	Specified Interrupt channel, start with 0. Please refer to the hardware manual for details.	
eddgetype	When the ISR will service the interrupt.	0 → active low 1 → active high
javaisr	Specified interrupt service routine.	

Return Value

0	No error.

See Also:open,close

removeInterrupt
public int **removeInterrupt**(int cardfd)

Remove the interrupt service routine that install at before.
Caution:The ISR must been remove before exit program otherwise maybe cause
system unstable.

Input Parameter

cardfd	File descriptor no of pci card that return from open.

Return Value

0	No error.

See Also:open,close

---

setCounter

public int **setCounter**(int cardfd,int counterno,int cicr,int mode,int value)

Set the 8254 counter's mode and value.

Input Parameter

cardfd	File descriptor no of pci card that return from open.
counterno	Specified counter number, start with 0.
cicr	Clock/Int Control register,Please refer to the hardware manual for details.
mode	The 8254 Counter-Mode: 0 to 5,Please refer to the hardware manual for details.
value	The 16 bits value for the timer/counter to count.

Return Value

0	No error.

See Also:open,close

3.2.3 Class IxpciDio

com.icpdas.cardio.ixpci

Class Ixpci

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpci.IxpciDio
```

Field Summary

u16, **mode**, **channel**

Field Detail

u16
public int **u16**
16 bit Digital input/output data.

mode
public int **mode**
Digital input/output mode.

channel
public int **channel**
Digital input/output channel, start with 0.

3.2.4 Class IxpciAio

com.icpdas.cardio.ixpci

Class Ixpci

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpci.IxpciAio
```

Field Summary

u16, **channel**

Field Detail

u16
public int **u16**
16 bit Analog input/output data.

channel
public int **channel**
Analog input/output channel, start with 0.

3.2.5 Class IxpciReg

com.icpdas.cardio.ixpci

Class Ixpci

java.lang.Object
```
  |
  +--com.icpdas.cardio.ixpci.IxpciReg
```

Field Summary
value, **mode**

Field Detail

value
public int **value**
The register value.

mode
public int **mode**
Register access mode.

3.2.6 Class IxpciInfo

com.icpdas.cardio.ixpci

Class Ixpci

java.lang.Object
```
  |
  +--com.icpdas.cardio.ixpci.IxpciInfo
```

Field Summary

no, **name**, **csid**, **irq**, **base**

Field Detail

no
public int **no**
The pci card identifier number in system.

name
public java.lang.String **name**
The pci card identifier name.

csid
public int **name**
The pci card hardware identifier number.

irq
public int **irq**
The pci card hardware interrupt request number.

base
public int **base**
The pci card hardware base address.

3.2.7 Hardware Register

PCI 1002 Register Summary	
IXPCI_PICR	PCI Interrupt Control Register.
IXPCI_8254C0	8254 Counter 0.
IXPCI_8254C1	8254 Counter 1.
IXPCI_8254C2	8254 Counter 2.
IXPCI_8254CR	8254 Control Register
IXPCI_AICR	Analog Input Channel Control Register.
IXPCI_SR	Status Register
IXPCI_AIGR	Analog Input Gain Control Register.
IXPCI_CR	Control Register.

IXPCI_ADST	AD Software Trigger.
IXPCI_DI	Digital Input.
IXPCI_DO	Digital Output.
IXPCI_AI	Analog Input (A/D Data Register).

PCI 1202,1602,1800,1802 Register Summary

IXPCI_8254C0	8254 Counter 0.
IXPCI_8254C1	8254 Counter 1.
IXPCI_8254C2	8254 Counter 2.
IXPCI_8254CR	8254 Control Register
IXPCI_CR	Control Register.
IXPCI_SR	Status Register
IXPCI_ADST	AD Software Trigger.
IXPCI_DI	Digital Input.
IXPCI_DO	Digital Output.
IXPCI_AD	Analog input.
IXPCI_DA1	Analog Output Channel 1.
IXPCI_DA2	Analog Output Channel 2.

PCI-P8R8/P16R16/P16C16 Register Summary

	There is no register available.

PCI TMC12 Register Summary

IXPCI_PICR	PCI Interrupt Control Register.
IXPCI_8254C0	8254 Counter 0.
IXPCI_8254C1	8254 Counter 1.
IXPCI_8254C2	8254 Counter 2.
IXPCI_8254CR	8254 Control Register
IXPCI_8254CS	A8254 Chip Select.
IXPCI_DI	Digital Input.
IXPCI_DO	Digital Output.

3.3 com.icpdas.ixpio Package

Package com.icpdas.ixpio Description:
The com.icpdas.ixpio package can be used to write platform-independent industry applications. Provides the classes necessary to support pio series add-on cards. ICPDAS pio add-on cards series is a family of high performance data acquisition board for PC with PCI bus.

(ICPDAS PIO-DA16 add-on cards)

Interface Summary	
IxpioIsr	Provide a interface to create a routin to handle a interrupt event.

Class Summary	
Ixpio	Defines methods that communication to pci series add-on cards.
IxpioDio	Digital input/output variable.
IxpioAio	Analog input/output variable.
IxpioReg	Register access variable.
IxpioInfo	Add-on cards peripheral information variable.

3.3.1Class IxpioIsr

com.icpdas.cardio.ixpio

Class IxpioIsr

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.IxpioIsr
```

Interrupt Service Routine (ISR) was a small routine that execute when interrupt happen. The IxpioIsr class Provide a interface to create a routine to handle a interrupt event.

Method Summary	
void	ixpio_isr(int sig) Called by JIOD to allow a routine to handle a interrupt request.

Method Detail

ixpio_isr
public void **ixpio_isr**(int sig)

The way to create a ISR is to declare a class that implements the IxpioIsr interface. That class then implements the ixpio_isr method. When the interrupt event occurs, that IxpioIsr's ixpio_isr method is invoked.

Example:

```
import java.io.*;                          //For System.in.read()
import com.icpdas.cardio.ixpio.*;          //ICPDAS PIO Series I/O Card packages

public class Interrupt implements IxpioIsr      //Implement IxpioIsr interrupt class
{
   public void ixpio_isr(int sig)               //Will be call when trigger interrupt
   {
      System.out.println("Java get signal " + sig);
   }

   public static void main(String[] args) throws java.io.IOException
   {
      int rev;
      int fd;
      Interrupt demo_isr;
      byte a[] = new byte[100];
      Ixpio icpdasio1=new Ixpio();
```

```
        fd=icpdasio1.open("ixpio2");
        if(fd==-1) System.out.println("Card Open Error Code:"+ fd);
        /* port configuration */
        if((rev=icpdasio1.setPort(fd,"IXPIO_PC",03))!=0)
                                    System.out.println("Configure PCA Error Code:"+ rev);
        /* Port 0/1 as DO, Port 2 as DI*/
        /* configure board interrupt */
        demo_isr = new Interrupt();
        /* INT_CHAN_0 */
        if((rev=icpdasio1.registerInterrupt(fd, 1, 0,demo_isr))!=0)
                                    System.out.println("Reg Int Error Code : "+ rev);
         /* INT_CHAN_0  signal for both edges */
        while(a[0]!=113)    /* read, get out if 'q' + Enter pressed */
        {
           System.in.read(a);
        }
        /*Remove Interrupt before program end*/
        if((rev=icpdasio1.removeInterrupt(fd))!=0) System.out.println("Remove Int Error Code : "+ rev);
        /* INT_CHAN_0  signal for both edges */
        icpdasio1.close(fd);
        System.out.println("End of program");
    }
}
```

See Also:open,close

3.3.2 Class Ixpio

com.icpdas.cardio.ixpio

Class Ixpio

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.Ixpio
```

The Ixpio Class provide pio series add-on cards control method. A pio series add-on cards can be opend for reading and writing data. Once the application is done with the port, it must call the close method before end program.

Method Summary	
int	open(java.lang.String filename) Initialize the pio card.
void	close(int cardfd) Free all the resources used by open. This method must be **called before** the program exit.

int	setPort(int cardfd, java.lang.String portid, int portattr) Configure I/O port attribute.
int	setReg(int cardfd, java.lang.String regname, IxpioReg IxpioArg) Write value to card register.
int	getReg(int cardfd, java.lang.String regname) Obtion register value of card.
int	getInfo(int cardfd, IxpioInfo IxpioArg) Obtion peripheral information of a card.
int	getDigitalIn(int cardfd, IxpioDio IxpioArg) Read the digital input value from card.
int	setDigitalOut(int cardfd, IxpioDio IxpioArg) To set the digital output value for card.
int	getAnalogIn(int cardfd, IxpioAio IxpioArg) Read the analog input value from card.
int	setAnalogOut(int cardfd, IxpioAio IxpioArg) Send the analog output command to card.
int	setVoltageOut(int cardfd, IxpioAio IxpioArg) Output the value of voltage (without the calibration) to the specified board and channel.
int	setCalVoltageOut(int cardfd, IxpioAio IxpioArg) Output the value of voltage to the specified board and channel.
int	setCurrentOut(int cardfd, IxpioAio IxpioArg) Output the value of current (without the calibration) to the specified board and channel.
int	setCalCurrentOut(int cardfd, IxpioAio IxpioArg) Output the value of current to the specified board and channel.
int	registerInterrupt(int cardfd,int interruptchannel,int eddgetype,IxpioIsr javaisr) Install user define interrupt service routine.
int	removeInterrupt(int cardfd) Remove user define interrupt service routine.
int	setCounter(int cardfd,int counterno,int cicr,int mode,int value) Configure counter on add-on card.

Method Detail

open

public int **open**(java.lang.String filename)

This method will open the card and allocate the resource for the device. This function must be called once before calling other method.

Input Parameter

filename	/dev/ixpion	The n is the PIO device number.

Return Value

fd	File descriptor number of pio card.

See Also:open,close

---

close

public void **close**(int cardfd)

This method will close the pio card and release the resource from the device. This method must be called once before exit the user's application.

Input Parameter

fd	File descriptor no of pio card that return from open.

See Also:open,close

---

setPort

public int **setPort**(int cardfd, java.lang.String portname, int portattr)

Configure specified I/O port attribute.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
portname	Name of the port control register.
portattr	The port setting value.

Return Value

0	No error.

See Also:open,close

---

setReg

public int **setReg**(int cardfd, java.lang.String regname, IxpioReg IxpioArg)

Write a given value to a specified register.

Input Parameter

cardfd	File descriptor no of pio card that return from open.

regname	Name of the register.

Return Value

0	No error.

See Also:open,close

---

getReg

public int **getReg**(int cardfd, java.lang.String regname)

Read a specified register value.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
regname	Name of the register.

Return Value

int	The register value.

See Also:open,close

---

getInfo

public int **getInfo**(int cardfd, IxpioInfo IxpioArg)

Get the I/O address of specified pio board.

Input Parameter

cardfd	File descriptor no of pio card that return from open.

Return Value

IxpioArg	Peripheral information of specified pio board.

Example:

```
import com.icpdas.cardio.ixpio.*;

public class List
{
   public static void main(String[] args)
   {
      int rev;
      int fd=0;
      int i=1;
      String devname = new String("ixpio");
      Ixpio icpdasio1=new Ixpio();
      IxpioInfo cardinfo = new IxpioInfo();
      while (fd!=-1)
      {
         fd=icpdasio1.open(devname + i);
         if (fd!=-1) //System.out.println("Open Card Error code: "+ fd);
```

```
            {
                rev=icpdasio1.getInfo(fd,cardinfo);
                if(rev==-1) System.out.println("Read Card Info Error code:"+ rev);
                System.out.println(devname + i +" CSID = "+ Integer.toHexString(cardinfo.csid));
                icpdasio1.close(fd);
            }
            i++;
        }
        System.out.println("End of program");
    }

}
```

See Also:open,close

getDigitalIn

public int **getDigitalIn**(int cardfd, IxpioDio IxpioArg)

This method will input the 16 bit data from the desired I/O port.

Input Parameter

cardfd	File descriptor no of pio card that return from open.

Return Value

IxpioArg.u16	Digital input value.
0	No error.

See Also:open,close

setDigitalOut

public int **setDigitalOut**(int cardfd, IxpioDio IxpioArg)

This method will send the 16 bits data to the desired I/O port.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.u16	Digital output value.

Return Value

0	No error.

See Also:open,close

getAnalogIn

public int **getAnalogIn**(int cardfd, IxpioAio IxpioArg)

Read the value of analog from the specified board and channel.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.u16	Analog input value.

Return Value

0	No error.

See Also:open,close

setAnalogOut

public int **setAnalogOut**(int cardfd, IxpioAio IxpioArg)

Output the value of analog to the specified board and channel.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.u16	Analog output value.

Return Value

0	No error.

See Also:open,close

setVoltageOut

public int **setVoltageOut**(int cardfd, IxpioAio IxpioArg)

This method will output the value of voltage (without the calibration) to the specified board and channel.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.channel	Specified Analog channel, start with 0.
IxpioArg.value	Voltage output value.

Return Value

0	No error.

See Also:open,close

setCalVoltageOut

public int **setCalVoltageOut**(int cardfd, IxpioAio IxpioArg)

Output the value of voltage to the specified board and channel. This function uses the EEPROM data to do the calibration.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.channel	Specified Analog channel, start with 0.
IxpioArg.value	Voltage output value.

Return Value

0	No error.

See Also:open,close

setCurrentOut
public int **setCurrentOut**(int cardfd, IxpioAio IxpioArg)

This method will output the value of current (without the calibration) to the specified board and channel.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.channel	Specified Analog channel, start with 0.
IxpioArg.value	Current output value.

Return Value

0	No error.

See Also:open,close

setCalCurrentOut
public int **setCalCurrentOut**(int cardfd, IxpioAio IxpioArg)

This method will output the value of current to the specified board and channel. This function uses the EEPROM data to do the calibration.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
IxpioArg.channel	Specified Analog channel, start with 0.
IxpioArg.value	Current output value.

Return Value

0	No error.

See Also:open,close

registerInterrupt
public int **registerInterrupt**(int cardfd,int interruptchannel,int eddgetype,IxpioIsr javaisr)

This method will install the interrupt service routine. This function supports multiple interrupt-source and the Active-Mode can setting to "Active-Low only", "Active-High only" and "Active-Low or Active-High".

Caution:The ISR method must use carefully and been remove before exit program otherwise maybe cause system unstable.

Input Parameter

cardfd	File descriptor no of pio card that return from open.	
interruptchannel	Specified Interrupt channel, start with 0. Please refer to the hardware manual for details.	
eddgetype	When the ISR will service the interrupt.	0 → active low 1 → active high
javaisr	Specified interrupt service routine.	

Return Value

0	No error.

See Also:open,close

removeInterrupt
public int **removeInterrupt**(int cardfd)

Remove the interrupt service routine that install at before.
Caution:The ISR must been remove before exit program otherwise maybe cause system unstable.

Input Parameter

cardfd	File descriptor no of pio card that return from open.

Return Value

0	No error.

See Also:open,close

setCounter
public int **setCounter**(int cardfd,int counterno,int cicr,int mode,int value)

Set the 8254 counter's mode and value.

Input Parameter

cardfd	File descriptor no of pio card that return from open.
counterno	Specified counter number, start with 0.
cicr	Clock/Int Control register,Please refer to the hardware manual for details.
mode	The 8254 Counter-Mode: 0 to 5,Please refer to the hardware manual for details.
value	The 16 bits value for the timer/counter to count.

Return Value

0	No error.

See Also:open,close

3.3.3 Class IxpioDio

com.icpdas.cardio.ixpio

Class Ixpio

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.IxpioDio
```

Field Summary
u16, **mode**, **channel**

Field Detail

u16
public int **u16**
16 bit Digital input/output data.

mode
public int **mode**
Digital input/output mode.

channel
public int **channel**
Digital input/output channel, start with 0.

3.3.4 Class IxpioAio

com.icpdas.cardio.ixpio

Class Ixpio

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.IxpioAio
```

Field Summary
u16, **channel**, **value**

Field Detail

u16
public int **u16**
16 bit Analog input/output data.

channel
public int **channel**
Analog input/output channel, start with 0.

value
public double **value**
Float Analog input/output value.

3.3.5 Class IxpioReg

com.icpdas.cardio.ixpio

Class Ixpio

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.IxpioReg
```

Field Summary
value, **mode**

Field Detail

value
public int **value**
The register value.

mode
public int **mode**
Register access mode.

3.3.6 Class IxpioInfo

com.icpdas.cardio.ixpio

Class Ixpio

```
java.lang.Object
  |
  +--com.icpdas.cardio.ixpio.IxpioInfo
```

Field Summary

no, **name**, **csid**, **irq**, **base**

Field Detail

no
public int **no**
The pio card identifier number in system.

name
public java.lang.String **name**
The pio card identifier name.

csid
public int **name**
The pio card hardware identifier number.

irq
public int **irq**
The pio card hardware interrupt request number.

base
public int **base**
The pio card hardware base address.

3.3.7 Hardware Register

PIO-D24 Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_Pn	Port n. The n is the port number. For example, the IXPIO_P0 represents the Port 0. The maximum value of the n is depended on the device (card) you are using, which can be found from the hardware manual.
IXPIO_PC[a]	Port Configuration [a]. The [a] is an option to specify the configuration-port. For example, the IXPIO_PCA generally represents the configuration for port 0 to 2, the IXPIO_PCB represents the configuration for port 3 to 5, and so on in the same rule. Omit the [a], the IXPIO_PC is identical to the IXPIO_PCA. Some devices (cards) have 3 configuration ports, some have only 1, which can be found from the hardware manual.

PIO-D48 Register Summary	
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.

IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_82551PA	8255 chip 1 port A
IXPIO_82551PB	8255 chip 1 port B
IXPIO_82551PC	8255 chip 1 port C
IXPIO_82551CW	8255 chip 1 control word
IXPIO_82552PA	8255 chip 2 port A
IXPIO_82552PB	8255 chip 2 port B
IXPIO_82552PC	8255 chip 2 port C
IXPIO_82552CW	8255 chip 2 control word
IXPIO_82541C0	8254 chip 1 counter 0
IXPIO_82541C1	8254 chip 1 counter 1
IXPIO_82541C2	8254 chip 1 counter 2
IXPIO_82541CW	8254 chip 1 control word

PIO-D56 Register Summary

IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_CON1L	Connector 1 low byte
IXPIO_CON1H	Connector 1 high byte
IXPIO_CON2L	Connector 2 low byte
IXPIO_CON2H	Connector 2 high byte
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital I/O

PIO-D64, PISO-A64/C64/P64 Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling condictions.
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital I/O
IXPIO_DIO_a	Digital IO group a. The a is the port group number, which generally represents an 8-bit digital port. For example, the IXPIO_DIO_A represents the DIO0-7, the IXPIO_DIO_B represents the DIO8-15, and so on in the same rule. Some devices (cards) have four groups (A, B, C, D), some haven't, which can be found from the hardware manual.
IXPIO_82541C0	8254 chip 1 counter 0
IXPIO_82541C1	8254 chip 1 counter 1
IXPIO_82541C2	8254 chip 1 counter 2
IXPIO_82541CW	8254 chip 1 control word
IXPIO_82542C0	8254 chip 2 counter 0
IXPIO_82542C1	8254 chip 2 counter 1
IXPIO_82542C2	8254 chip 2 counter 2
IXPIO_82542CW	8254 chip 2 control word

PIO-D96 Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register

IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_Pn	Port n. The n is the port number. For example, the IXPIO_P0 represents the Port 0. The maximum value of the n is depended on the device (card) you are using, which can be found from the hardware manual.
IXPIO_PC[a]	Port Configuration [a]. The [a] is an option to specify the configuration-port. For example, the IXPIO_PCA generally represents the configuration for port 0 to 2, the IXPIO_PCB represents the configuration for port 3 to 5, and so on in the same rule. Omit the [a], the IXPIO_PC is identical to the IXPIO_PCA. Some devices (cards) have 3 configuration ports, some have only 1, which can be found from the hardware manual.

PIO-D144 Register Summary

IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_8DR	8-bit Register
IXPIO_AIOPCR	Active I/O Port Control Register
IXPIO_IOSCRA	I/O Select Control Register A
IXPIO_IOSCRB	I/O Select Control Register B
IXPIO_IOSCRC	I/O Select Control Register C

PIO-DA16/DA8/DA4 Register Summary

IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling condictions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling condictions.
IXPIO_82541C0	8254 chip 1 counter 0
IXPIO_82541C1	8254 chip 1 counter 1
IXPIO_82541C2	8254 chip 1 counter 2
IXPIO_82541CW	8254 chip 1 control word
IXPIO_DAnCS	The on board DAC chip select (W). The n is the DAC chip number. For example, IXPIO_DA0CS, IXPIO_DA1CS, IXPIO_DA2CS, IXPIO_DA3CS to represent the four DAC chips. The number of the DAC chips can be found from the hardware manual.
IXPIO_DAL	Analog output, low byte.
IXPIO_DAH	Analog output, high byte.
IXPIO_DIO_L	Digital IO, low byte
IXPIO_DIO_H	Digital IO, high byte
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital IO

PISO-725 Register Summary

IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of

	directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs

PISO-730/730A Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital IO
IXPIO_DIO_L	Digital IO, low byte
IXPIO_DIO_H	Digital IO, high byte
IXPIO_IDIO	The whole isolated digital IO
IXPIO_IDIO_L	Isolated digital IO, low byte
IXPIO_IDIO_H	Isolated digital IO, high byte

PISO-813 Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ADL	Analog input, low byte.
IXPIO_ADH	Analog input, high byte.
IXPIO_AD	Analog Input
IXPIO_MCSR	Multiplexer Channel Select Register

IXPIO_PGCR	PGA Gain Code Register
IXPIO_ADTCR	AD Trigger Control Register

PISO-P32C32 Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_ACR	AUX Control Register
IXPIO_ADR	AUX Data Register
IXPIO_IMCR	INT Mask Control Register
IXPIO_ASR	AUX Pin Status Register (R/W) The ASR is directly handled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_IPCR	Interrupt Polarity Control Register (R/W) The IPCR is directly heandled by device driver. Write to the register is not recommended! Instead of directly write to it, use the IXPIO_SIG ioctl command to set the interrupt signaling conditions.
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital I/O
IXPIO_DIO_a	Digital IO group a. The a is the port group number, which generally represents an 8-bit digital port. For example, the IXPIO_DIO_A represents the DIO0-7, the IXPIO_DIO_B represents the DIO8-15, and so on in the same rule. Some devices (cards) have four groups (A, B, C, D), some haven't, which can be found from the hardware manual.

PISO-P8R8/P8SSR8x Register Summary	
IXPIO_RCR	Reset Control Register
IXPIO_DI	The whole digital inputs
IXPIO_DO	The whole digital outputs
IXPIO_DIO	The whole digital I/O