Introduction to the Java Device I/O APIs

Jen Dority
Senior Member of Technical Staff
October 1, 2014

Program Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

Program Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

The Device I/O Project

The Device I/O Project is an OpenJDK to provide a Java-level API for accessing generic device peripherals on embedded devices.

The Device I/O Project

- Follows the JavaME Device I/O API
- Targets Linux/ARM SBCs
  - Raspberry Pi
  - SABRE Lite
- Supports an initial set of four peripheral device APIs
  - GPIO
  - SPI
  - I2C
  - UART

The Device I/O Project

(continued)

- Provides a consistent method for accessing low level peripherals on embedded devices
- Is extendable with service providers
- Helps developers manage multiple hardware configurations by providing the ability to assign logical names to devices

Program Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

Building The Device I/O Libraries

- Supports building on a Linux host with ARM cross-compiler
- Requires JDK7 or JDK8, Linux/ARM cross-compiler and GNU Make
- Sample code may also use the Ant build tool

Building The Device I/O Libraries

- Define required environment variables
  - `export JAVA_HOME=<path to JDK>`
  - `export PI_TOOLS=<path to Linux/ARM cross-compiler>`
- Get the source
  - `hg clone http://hg.openjdk.java.net/dio/dev`
- Build
  - `cd dev`
  - `make`

Building The Device I/O Libraries

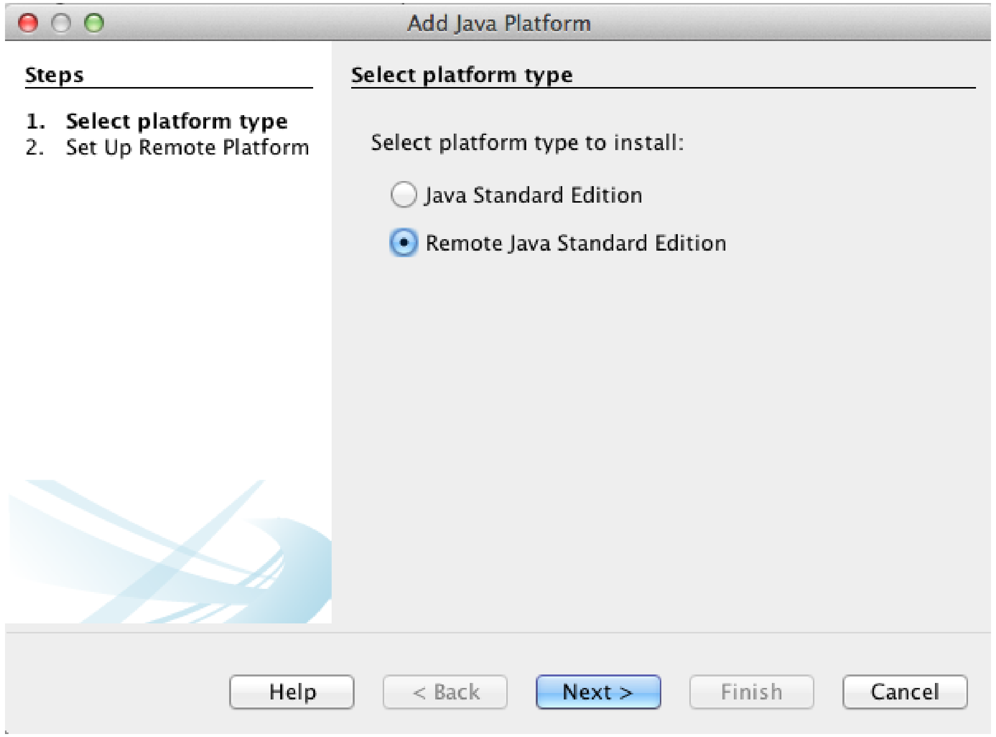
(continued)

- Completed library files will be in build directory
  - `<top-level>/build/jar/dio.jar`
  - `<top-level>/build/so/libido.so`

Program Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

Working With DIO APIs in Netbeans



Steps

1. **Select platform type**
2. Set Up Remote Platform

Select platform type

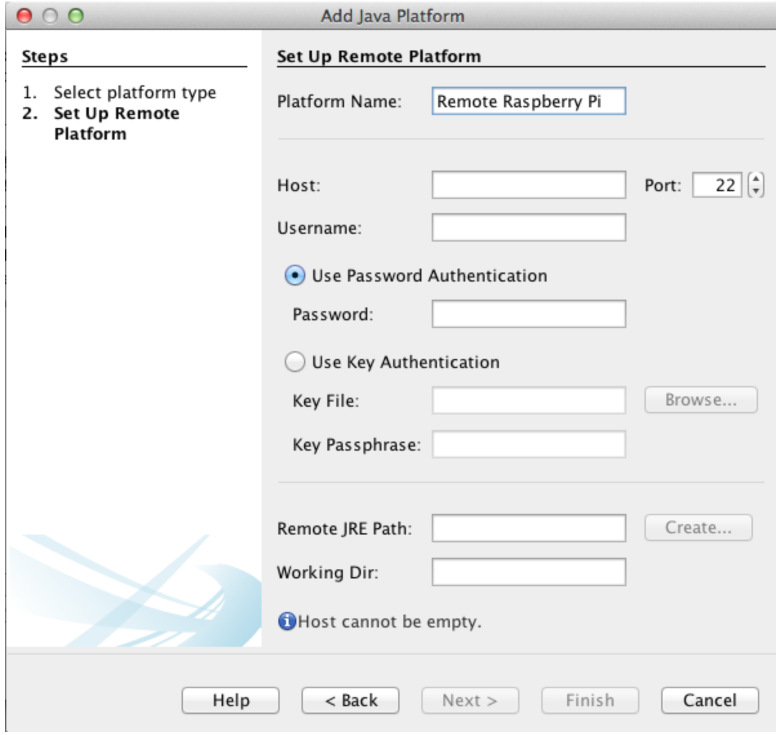
Select platform type to install:

○ Java Standard Edition

● Remote Java Standard Edition

Tools → Java Platforms → Add Platform . . .

Select “Remote Java Standard Edition” then click next



Steps

1. Select platform type
2. **Set Up Remote Platform**

Set Up Remote Platform

Platform Name: Remote Raspberry Pi

Host: Port: 22

Username:

● Use Password Authentication

Password:

○ Use Key Authentication

Key File:

Key Passphrase:

Remote JRE Path:

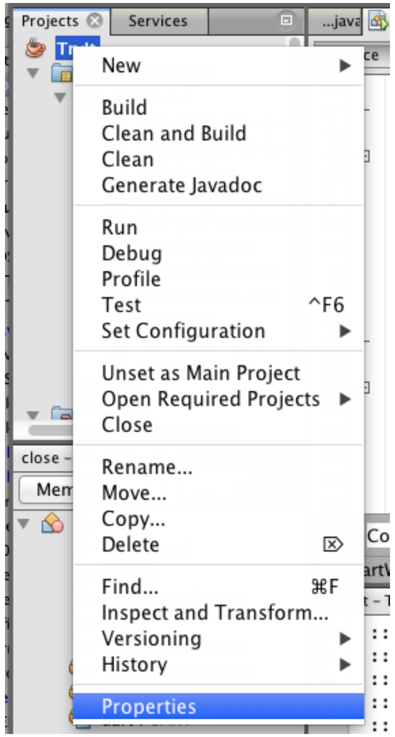
Working Dir:

Host cannot be empty.

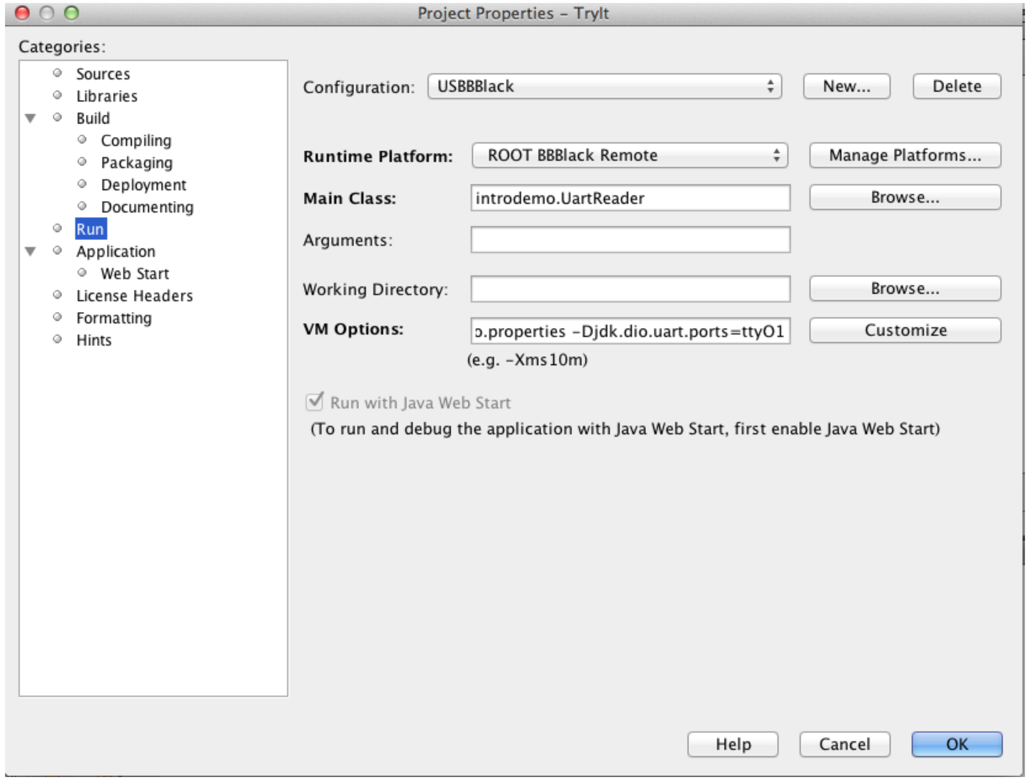
Fill in required fields then click “Finish”

Note: may need to use “root” credentials to run DIO apps from netbeans

Working With DIO APIs in Netbeans (cont’d)



Right click on your project and select “Properties”


Project Properties – TryIt

Configuration: USBBBlack

Runtime Platform: ROOT BBBlack Remote

Main Class: introdemo.UartReader

Arguments:

Working Directory:

VM Options: ɔ.properties –Djdk.dio.uart.ports=ttyO1

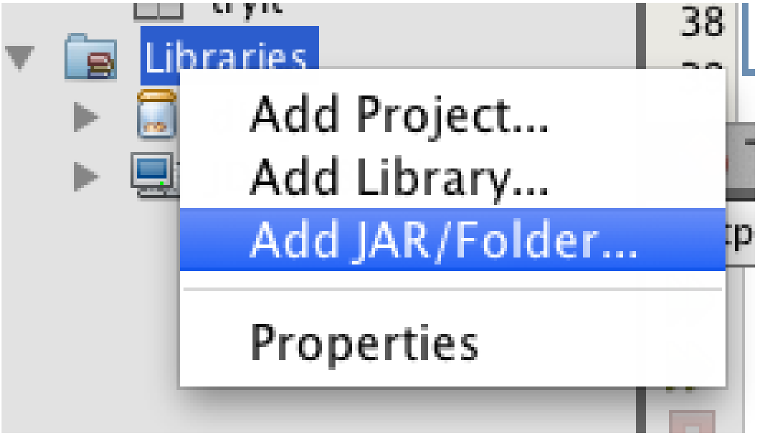
(e.g. –Xms10m)

☑ Run with Java Web Start

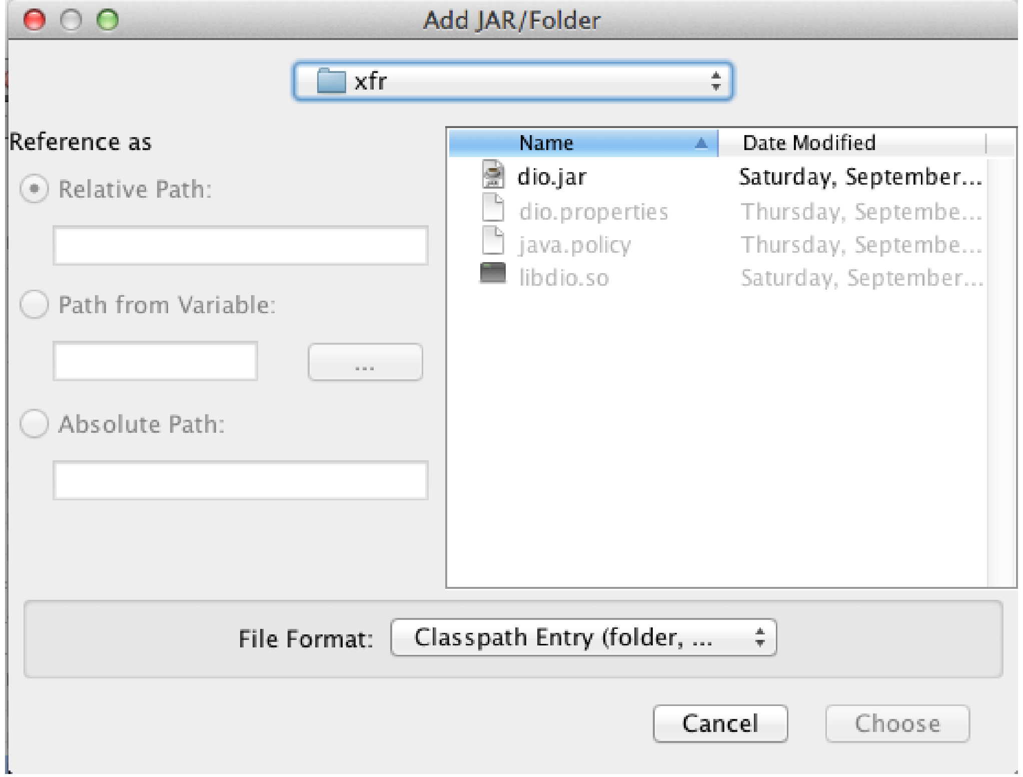
(To run and debug the application with Java Web Start, first enable Java Web Start)

Create a new configuration with your new remote platform

Working With DIO APIs in Netbeans (cont’d)



Right click on “libraries” in your project tree and select “Add JAR/Folder…”



Reference as

● Relative Path:

○ Path from Variable:

○ Absolute Path:

Name	Date Modified
dio.jar	Saturday, September...
dio.properties	Thursday, Septembe...
java.policy	Thursday, Septembe...
libdio.so	Saturday, September...

File Format: Classpath Entry (folder, ...

Choose the dio.jar file

Using the Device I/O APIs

- Copy libido.so to your native java library path on the target device
  - Or, specify its location with –Djava.library.path in VM options
- Specify -Djdk.dio.registry in VM options (or in the `java` command line) to use a `.properties` file to preload a set of device configurations which you can refer to by a numeric ID
- Use DeviceManager.list() to get a list of all preloaded and user-registered devices in the system
- Get a device instance by using DeviceManager.open() methods
- When done with a device, be sure to call its close() method
- Access to devices depends on appropriate OS level access and new Java permissions

Program Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

A Closer Look . . .

GPIO

GPIO

- General Purpose Input/Output
- Logical 1 or 0 controlled by software
- Dedicated to a single purpose
  - Drive a single LED
  - Status flag
  - “bit-banging”

jdk.dio.gpio.GPIOPin

Key configuration details

- Pin number
- Direction
  - Input
  - Output
- Trigger
  - Rising
  - Falling
- Mode – Not software configurable for Linux/ARM port

jdk.dio.gpio.GPIOPin

- Represents a single GPIO pin
- Can be configured as input or output
  - Detect a button press
  - Drive a single LED
- Can register listeners to handle “value changed” events

jdk.dio.gpio.GPIOPin

```
GPIOPinConfig config =
               new GPIOPinConfig(DeviceConfig.DEFAULT,     // controller number
                                 18,                       // pin number
                                 GPIOPinConfig.DIR_OUTPUT_ONLY,
                                 GPIOPinConfig.DEFAULT,    // mode (ignored)
                                 GPIOPinConfig.TRIGGER_NONE,
                                 false);                   // initial value

 . . .
       GPIOPin outputPin = DeviceManager(config);
 . . .
       outputPin.setValue(true);
```

jdk.dio.gpio.GPIOPin

```
GPIOPinConfig config =
               new GPIOPinConfig(DeviceConfig.DEFAULT,
                                 23,                                 // pin number
                                 GPIOPinConfig.DIR_INPUT_ONLY,
                                 GPIOPinConfig.DEFAULT,
                                 GPIOPinConfig.TRIGGER_RISING_EDGE |
                                         GPIOPinConfig.TRIGGER_FALLING_EDGE,
                                 false);                             // initial value

 . . .
       GPIOPin inputPin = DeviceManager(config);
 . . .
       boolean pinValue = inputPin.getValue();
```

jdk.dio.gpio.GPIOPin

```
inputPin.setInputListener(new PinListener() {
        public void valueChanged(PinEvent event) {
                System.out.println(“Pin value is now “ + event.getValue());
        }
});
```

A Closer Look . . .

SPI

SPI

- Serial Peripheral Interface
- Single master/multiple slaves connected to a single bus
- Serial, full-duplex
- Bits shift in on MISO (Master In Slave Out) as they shift out on MOSI (Master Out Slave In)

jdk.dio.spibus.SPIDevice

Key configuration details

- Device number
- Chip select address (device address)
- Chip select active level
  - High, low, not controlled
- Clock mode – see javadocs for explanation
- Word length
- Bit ordering

jdk.dio.spibus.SPIDevice

- Represents an SPI slave device
- Provides methods to write, read and writeAndRead to/from the slave device
  - write(); read(); != writeAndRead();
- Allows you to surround a series of writes and reads with begin(), end() to keep slave select line active
- Uses java.nio.ByteBuffer in API calls

jdk.dio.spibus.SPIDevice

```
SPIDeviceConfig config =
               new SPIDeviceConfig(DeviceConfig.DEFAULT,          // Device Number
                                                      0,          // SS connected to CE0
                                                      500000,     // clock frequency
                                                      SPIDeviceConfig.CS_ACTIVE_LOW,
                                                      8,          // 8-bit words
                                                      Device.BIG_ENDIAN);
 . . .
SPIDevice spiDevice = DeviceManager.open(config);
 . . .
```

jdk.dio.spibus.SPIDevice

MCP3008 Example

```
public int readChannel(int c) {
        ByteBuffer out = ByteBuffer.allocate(3);
        ByteBuffer in = ByteBuffer.allocate(3);
        out.put((byte)0x01);                              // start bit
        out.put((byte)(((c | 0x08) & 0x0f) << 4));        // single-ended, channel c
        out.put((byte)0);                                 // padding
        out.flip();      // important!!! reset or flip buffer to start sending from

                         // the beginning
        . . .
            spiDevice.writeAndRead(out, in);
        . . .
        int high = (int)(0x0003 & in.get(1));    // first byte is padding, 10-bit result is
        int low = (int)(0x00ff & in.get(2));      // contained in bit 1-0 of second byte and
                                                  // all eight bits of third byte
        return (high << 8) + low;
    }
```

A Closer Look . . .

I^{2}C

I^{2}C

- Inter-Integrated Circuit
- Multi-master/multi-slave bus
  - Device I/O supports only slave devices
  - One master is assumed
- Serial, half-duplex
- One line for data, one for clock, no separate address lines

jdk.dio.i2cbus.I2CDevice

Key configuration details

- Controller number
- Slave address
- Address size
- Clock frequency

jdk.dio.i2cbus.I2CDevice

- Represents a I2C slave device
- Provides methods to read, write from/to slave device
- Allows you to surround a series of related writes and reads with begin(), end()
- Uses java.nio.ByteBuffer in API calls

jdk.dio.i2cbus.I2CDevice

BMP160 Example

```
I2CDeviceConfig config =
                new I2CDeviceConfig(1,           // i2c bus number (raspberry pi)
                                    0x77,        // i2c slave address (BMP180 press/temp sensor)
                                    7,           // address size in bits
                                    3400000);    // 3.4MHz clock frequency


        . . .
I2CDevice i2cSlave = DeviceManager.open(config);
        . . .
// read calibration data
ByteBuffer dst = ByteBuffer.allocate(22);// 22 = size (bytes) of calibration data
int bytesRead = i2cSlave.read(0xAA,      // EEPROM start address
                              1,         // size (bytes) of subaddress
                              dst);
```

A Closer Look . . .

UART

UART

- Universal Asynchronous Receiver/Transmitter

jdk.dio.uart.UART

Key configuration details

- Controller name or number
- Baud rate
- Parity
- Stop bits
- Flow control

jdk.dio.uart.UART

- Allows for control and access of a UART device
- Provides methods to for synchronous and asynchronous reads and writes
- Implements the java.nio.channels interfaces ReadableByteChannel and WriteableByteChannel
- Uses java.nio.ByteBuffer in API calls

jdk.dio.uart.UART

```
UARTConfig config  = new UARTConfig(“ttyAMA0”,            // device name
                                    DeviceConfig.DEFAULT, // channel
                                    9600,                 // baud rate
                                    UARTConfig.DATABITS_7,
                                    UARTConfig.PARITY_NONE,
                                    UARTConfig.FLOWCONTROL_NONE);
        . . .
UART uart = DeviceManager.open(config);
OutputStream os = Channels.newOutputStream(uart);
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os));

writer.(“Hello”);

        . . .
```

More Info

- Device I/O OpenJDK Project page
  - http://openjdk.java.net/projects/dio/
- Device I/O mailing list
  - http://mail.openjdk.java.net/mailman/listinfo/dio-dev
- Device I/O Wiki
  - https://wiki.openjdk.java.net/display/dio/Main
- Device I/O mercurial repo
  - http://hg.openjdk.java.net/dio/dev

Safe Harbor Statement

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

JavaOne™

ORACLE®

ORACLE®