7 General Purpose Input/Output (GPIO)

This chapter discusses the interfaces and classes for reading from and writing to the General Purpose Input/Output (GPIO) pins and ports of the embedded device board.

A GPIO pin is a generic pin whose value consists of one of two voltage settings (*high* or *low*) and whose behavior can be programmed through software. A GPIO port is a platform-defined grouping of GPIO pins (often 4 or more pins). However, GPIO pins that are part of a GPIO port cannot be retrieved or controlled individually as GPIO pins.

In order to use a specific pin or port, an application should first open and obtain a `GPIOPin` or `GPIOPort` instance for the pin or port it wants to use, using its numerical ID, name, type (interface), or properties.

Here is an example of obtaining a `GPIOPin` and a `GPIOPort` using its ID:

```
  GPIOPin pin = (GPIOPin) PeripheralManager.open(1);
  GPIOPort port = (GPIOPort) PeripheralManager.open(0);
```

Here is an example of using its name and interface:

```
  GPIOPin pin = (GPIOPin) PeripheralManager.open("LED_PIN", GPIOPin.class, null);
  GPIOPort port = (GPIOPort) PeripheralManager.open("LCD_DATA_PORT",
      GPIOPort.class, null);
```

Once a pin is opened, an application can obtain the current value of a GPIO pin by calling the `GPIOPin.getValue()` method and set its value by calling the `GPIOPin.setValue(boolean)` method. Likewise, once a port opened, an application can obtain the current value of a GPIO port by calling the `GPIOPort.getValue()` method and set its value by calling the `GPIOPort.setValue(int)` method.

```
  pin.setValue(true);
  port.setValue(0xFF);
```

When done, the application should call the `GPIOPin.close()` or `GPIOPort.close()` method to release the pin or port, respectively.

```
  pin.close();
  port.close();
```

Example 7-1 gives an demonstration of using the GPIO API. First, it registers a pin listener for the GPIO input pin that a switch button is attached to. When the button is pressed, the listener is notified. The listener then turns the LED on or off by setting the GPIO output pin that the LED is attached to accordingly.

Example 7-1 Using the GPIO APIs

```
import com.oracle.deviceaccess.PeripheralManager;
import com.oracle.deviceaccess.PeripheralNotAvailableException;
import com.oracle.deviceaccess.PeripheralNotFoundException;
import com.oracle.deviceaccess.gpio.GPIOPin;
import com.oracle.deviceaccess.gpio.PinEvent;
import com.oracle.deviceaccess.gpio.PinListener;
import java.io.IOException;
```

```
public class GPIODemo {

    GPIOPin switchPin = null;
    GPIOPin ledPin = null;

    public GPIODemo() {
        try {
            switchPin = (GPIOPin) PeripheralManager.open(1);
            ledPin = (GPIOPin) PeripheralManager.open(3);
            switchPin.setInputListener(new PinListener() {
                public void valueChanged(PinEvent event) {
                    try {
                        ((GPIOPin) event.getPeripheral()).
                            setValue(event.getValue()); // turn LED on or off
                    } catch (IOException ex) {
                        // Ignored
                    } catch (PeripheralNotAvailableException ex) {
                        // Ignored
                    }
                }
            });
        } catch (IOException ex) {
            // Handle exception
        } catch (PeripheralNotFoundException ex) {
            // Handle exception
        } catch (PeripheralNotAvailableException ex) {
            // Handle exception
        } finally {
            if (switchPin != null) {
                try {
                    switchPin.close();
                } catch (IOException ex) {
                }
            }
            if (ledPin != null) {
                try {
                    ledPin.close();
                } catch (IOException ex) {
                }
            }
        }
    }
}
```

Note that the underlying platform configuration may allow for some GPIO pins or ports to be set by an application for either output or input, while others may be used for input only or output only and their direction cannot be changed by an application. Note also that asynchronous notification of pin or port value changes is only loosely tied to hardware-level interrupt requests. The platform does not guarantee notification in a deterministic or timely manner.

Because of performance issue, procedures handling GPIO pins, and especially event listeners, should be implemented to be as fast as possible.

GPIO pins and ports are opened by invoking one of the `com.oracle.deviceaccess.PeripheralManager.open()` methods. The permissions in Table 7-1 allow access to be granted to GPIO pins and ports. as a whole as well as to some of their protected functions. These permissions must be requested in the JAD file under `MIDlet-Permissions` or `MIDlet-Permissions-Opt`, and the application must be digitally signed by a trusted authority to gain access to the APIs. Alternatively, the permissions may be allowed for all applications in the `untrusted` domain of the security policy file (`policy.txt`).

Table 7-1 GPIO API Permissions

Permission	Description
`com.oracle.deviceaccess.gpio`	Access to GPIO pins and ports (as a whole)
`com.oracle.deviceaccess.gpio.GPIOPin.setDirection`	Changing the direction of a GPIO pin
`com.oracle.deviceaccess.gpio.GPIOPort.setDirection`	Changing the direction of a GPIO port

The GPIOPin Interface

The `GPIOPin` interface provides methods for controlling a GPIO pin. A GPIO pin can be configured for output or input. Output pins are both writable and readable while input pins are only readable. The interface contains two constants, as shown in Table 7-2:

Table 7-2 GPIOPin Direction Constants

Constant	Description
`GPIOPin.INPUT`	The GPIO pin is configured for input and is only readable.
`GPIOPin.OUTPUT`	The GPIO pin is configured for output and is both readable and writable.

Each GPIO pin is identified by a numerical ID and by a name. A `GPIOPin` instance can be opened by a call to one of the `PeripheralManager.open()` methods. Once opened, an application can obtain the current value of a GPIO pin by calling the `getValue()` method and set its value by calling the `setValue(boolean)` method.

An application can either monitor a GPIO pin value changes using polling or can register a `PinListener` instance, which will be asynchronously notified of any pin value changes. To register a `PinListener` instance, the application must call the `setInputListener(PinListener)` method. The registered listener can later on be removed by calling the same method with a null listener parameter. Asynchronous notification is only supported for GPIO pins configured for input. An attempt to set a listener on a GPIO pin configured for output will throw an `InvalidOperationException`.

When an application is no longer using a GPIO pin, it should call the `GPIOPin.close()` method to release the GPIO pin. Any further attempt to set or get the value of a GPIO pin which has been closed will throw a `PeripheralNotAvailableException`.

The initial direction of a GPIO pin which may be used for output or input. The initial value of a GPIO pin set for output is configuration-specific. An application should always initially set the GPIO pin's direction; or first query the GPIO pin's direction then set it if necessary.

Note that the configuration may allow for some GPIO pins to be set by the application for either output or input, while others may be used for input only or output only and their direction cannot be changed by the application. Note also that asynchronous notification of pin value changes is only loosely tied to hardware-level interrupt requests. The platform does not guarantee notification in a deterministic or timely manner.

The `GPIOPin` interface consists of the following methods:

- `int getDirection() throws IOException, PeripheralNotAvailableException`

  This method returns the current pin direction: `GPIOPin.OUTPUT` if this GPIO pin is currently set as output, or `GPIOPin.INPUT` if it is set as input.

- `boolean getValue() throws SecurityException, IOException, PeripheralNotAvailableException`

This method returns the current value of the GPIO pin. This method can be called on both output and input pins. This method returns `true` if this pin is currently high, or `false` if it is low.

- `void setDirection(int direction) throws IOException, PeripheralNotAvailableException`

  This methods sets the GPIO pin direction, either for output or input. Any attempt to set a GPIO pin to a direction not supported by the platform configuration throws an `InvalidOperationException`.

- `void setValue(boolean value) throws IOException, PeripheralNotAvailableException`

  This method sets the value of this GPIO pin. The boolean parameter represents the new pin value: `true` for high, `false` for low. Any attempt to set the value on a GPIO pin currently not configured for output throws an `InvalidOperationException`.

- `void setInputListener(PinListener listener) throws java.io.IOException, PeripheralNotAvailableException`

  This method registers a `PinListener` instance which will get asynchronously notified when this GPIO pin's value changes and according to the current trigger mode (see `GPIOPinConfig.getTrigger()`). Notification will automatically begin after registration completes. A listener can only be registered for a GPIO pin currently configured for input, and only one listener can be registered at a time. If the parameter passed in is `null`, the current listener is removed.

The GPIOPort Interface

The `GPIOPort` interface provides methods for controlling a GPIO port. A GPIO port is a platform-defined grouping of GPIO pins that can be configured for output or input. Like GPIO pins, each GPIO port is identified by a numerical ID and by a name. Output ports are both writable and readable while input ports are only readable. GPIO pins that are part of a GPIO port cannot be retrieved or controlled as individual `GPIOPin` instances.

The `GPIOPort` interface contains two constants, as shown in Table 7-3:

Table 7-3 GPIOPort Direction Constants

Constant	Description
`GPIOPort.INPUT`	The GPIO port is configured for input, and is only readable.
`GPIOPort.OUTPUT`	The GPIO port is configured for output, and is both readable and writable.

A `GPIOPort` instance can be opened by a call to one of the `PeripheralManager.open()` methods. Once opened, an application can obtain the current value of a GPIO port by calling the `getValue()` method and set its value by calling the `setValue(int)` method. A GPIO port has a minimum and maximum value range. The minimum value is zero. An application can check the maximum value by calling `getMaxValue()` method. An attempt to set a GPIO port with a value that exceeds its maximum range value will throw an `IllegalArgumentException`.

An application can either monitor a GPIO port value changes using polling or can register a `PortListener` instance, which will be asynchronously notified of any value changes. To register a `PortListener` instance, the application must call the `setInputListener(PortListener)` method. The registered listener can later on be removed by calling the same method with a null listener parameter. Asynchronous notification is only supported for GPIO port configured for input. An attempt to set a listener on a GPIO port configured for output will throw an `InvalidOperationException`.

When an application is no longer using a GPIO port, it should call the `GPIOPort.close()` method to release the GPIO port. Any further attempt to set or get the value of a GPIO port which has been closed will throw a `PeripheralNotAvailableException`.

The initial direction of a GPIO port which may be used for output or input. The initial value of a GPIO port set for output is configuration-specific. An application should always initially set the GPIO port's direction; or first query the GPIO port's direction then set it if necessary.

Note that the configuration may allow for some GPIO ports to be set by an application for either output or input, while others may be used for input only or output only and their direction cannot be changed by an application. Note also that asynchronous notification of port value changes is only loosely tied to hardware-level interrupt requests. The platform does not guarantee notification in a deterministic or timely manner.

The `GPIOPort` interface consists of the following methods:

- `int getDirection() throws IOException, PeripheralNotAvailableException`

  This method returns the current port direction: `GPIOPort.OUTPUT` if this GPIO port is currently set as output, or `GPIOPort.INPUT` if the port is set as input.

- `int getMaxValue() throws IOException, PeripheralNotAvailableException`

  This method returns the maximum value of this GPIO port. The value returned should be interpreted as an unsigned 32-bit integer.

- `int getValue() throws SecurityException, IOException, PeripheralNotAvailableException`

  This method returns the current value of this GPIO port. The value returned is interpreted as an unsigned 32-bit integer, and depending on the platform configuration, the value of each pin can be tested against a bit in the resulting value. This method can be called on both output and input pins.

- `void setDirection(int direction) throws IOException, PeripheralNotAvailableException`

  This method sets the GPIO port for output or input. Any attempt to set the direction of a GPIO port to a value that is not supported by the platform configuration throws an `InvalidOperationException`.

- `void setValue(int value) throws IOException, PeripheralNotAvailableException`

  This method sets the value of this GPIO port. Any attempt to set the value on a GPIO port currently not configured for output throws an `InvalidOperationException`. The value passed is interpreted as an unsigned 32-bit integer.

- `void setInputListener(PortListener listener) throws java.io.IOException, PeripheralNotAvailableException`

  This method registers a `PortListener` instance that is asynchronously notified when this GPIO port's value changes. Notification automatically begins after registration completes. A listener can only be registered for a GPIO port currently configured for input, and only one listener can be registered at a time. If the parameter is `null`, the current listener is removed.

The PinListener Interface

The `PinListener` interface provides a means of notification if a GPIO pin value changes. A `PinListener` can be registered using the `GPIOPin.setInputListener(com.oracle.deviceaccess.gpio.PinListener)` method.

The interface consists of only one method, `void valueChanged(PinEvent event)`, which is invoked when a GPIO pin's value has changed.

The PortListener Interface

The `PortListener` interface provides a means of notification if a GPIO port value changes. A `PortListener` can be registered using the `GPIOPort.setInputListener(com.oracle.deviceaccess.gpio.PortListener)` method.

The interface consists of only one method, `void valueChanged(PortEvent event)`, which is invoked when a GPIO port's value has changed.

The GPIOPinConfig Class

The `GPIOPinConfig` class encapsulates the configuration parameters of a GPIO pin. An instance of `GPIOPinConfig` can be passed to the `PeripheralManager.open(PeripheralConfig)` method to open the designated GPIO pin with the specified configuration.

The `GPIOPinConfig` class consists of several constants. The first four represent possible directions for the GPIO pin, and are shown inTable 7-4.

Table 7-4 Direction Constants in the GPIOPinConfig Class

Constant	Description
`DIR_INPUT_ONLY`	Input direction
`DIR_OUTPUT_ONLY`	Output direction
`DIR_BOTH_INIT_INPUT`	Bidirectional with initial input direction.
`DIR_BOTH_INIT_OUTPUT`	Bidirectional with initial output direction.

The next four are possible values for the mode, and are shown in Table 7-5. Note that the mode can also be `PeripheralConfig.DEFAULT`.

Table 7-5 Mode Constants in the GPIOPinConfig Class

Constant	Description
`MODE_INPUT_PULL_UP`	Input pull-up drive mode.
`MODE_INPUT_PULL_DOWN`	Input pull-down drive mode.
`MODE_OUTPUT_PUSH_PULL`	Output push-pull drive mode.
`MODE_OUTPUT_OPEN_DRAIN`	Output open-drain drive mode.

Finally, the last seven are possible values for the trigger, and are shown in Table 7-6.

Table 7-6 Trigger Constants in the GPIOPinConfig Class

Constant	Description
`TRIGGER_NONE`	No interrupt trigger.
`TRIGGER_FALLING_EDGE`	Falling edge trigger.
`TRIGGER_RISING_EDGE`	Rising edge trigger.
`TRIGGER_BOTH_EDGES`	Both edges trigger.
`TRIGGER_HIGH_LEVEL`	High level trigger.
`TRIGGER_LOW_LEVEL`	Low level trigger.
`TRIGGER_BOTH_LEVELS`	Both levels trigger.

The `GPIOPinConfig` class consists of one constructor and six methods:

- `public GPIOPinConfig(int portNumber, int pinNumber, int direction, int mode, int trigger, boolean initValue)`

  This constructor creates a new `GPIOPinConfig` with the provided parameters. See the earlier discussion for possible constant values for `direction`, `mode`, and `trigger`.

- `public int getDirection()`

  This method returns the configured pin direction. The pin direction can be one of: `DIR_INPUT_ONLY`, `DIR_OUTPUT_ONLY`, `DIR_BOTH_INIT_INPUT`, or `DIR_BOTH_INIT_OUTPUT`.

- `public boolean getInitValue()`

  This method returns the configured initial boolean value of the pin, if configured for output.

- `public int getPortNumber()`

  This method returns the configured port number for the pin.

- `public int getPinNumber()`

  This method returns the configured pin number.

- `public int getDriveMode()`

  This method returns the configured pin drive mode. The possible values can be: either `PeripheralConfig.DEFAULT` or a bitwise OR of at least one of : `MODE_INPUT_PULL_UP`, `MODE_INPUT_PULL_DOWN`, `MODE_OUTPUT_PUSH_PULL`, and `MODE_OUTPUT_OPEN_DRAIN`.

- `public int getTrigger()`

  This method returns the configured pin interrupt trigger. The pin interrupt trigger can be one of: `TRIGGER_NONE`, `TRIGGER_FALLING_EDGE`, `TRIGGER_RISING_EDGE`, `TRIGGER_BOTH_EDGES`, `TRIGGER_HIGH_LEVEL`, `TRIGGER_LOW_LEVEL`, `TRIGGER_BOTH_LEVELS`.

The GPIOPortConfig Class

The `GPIOPortConfig` class encapsulates the configuration parameters of a GPIO port. An instance of `GPIOPortConfig` can be passed to the `PeripheralManager.open(PeripheralConfig)` method to open the designated GPIO port with the specified configuration. Note that the interrupt trigger of a GPIO port is defined by the interrupt triggers configured for its pins. For more information, see `GPIOPinConfig.getTrigger()`.

The `GPIOPortConfig` class contains four constants, representative of the direction of the port. The constants are shown in Table 7-7.

Table 7-7 Direction Constants in the GPIOPortConfig Class

Constant	Description
`DIR_INPUT_ONLY`	Input port direction.
`DIR_OUTPUT_ONLY`	Output port direction.
`DIR_BOTH_INIT_INPUT`	Bidirectional port direction with initial input direction.
`DIR_BOTH_INIT_OUTPUT`	Bidirectional port direction with initial output direction.

The `GPIOPortConfig` class consists of one constructor and three methods:

- `GPIOPortConfig(int direction, int initValue, GPIOPinConfig[] pins)`

  This constructor creates a new `GPIOPortConfig` with the provided parameters. See the earlier discussion for possible constant values for `direction`.

- `public int getDirection()`

  This method returns the configured pin direction.

- `public boolean getInitValue()`

  This method returns the configured initial boolean value of the pin, if configured for output.

- `public GPIOPinConfig[] getPins()`

  This method returns the configured pins composing the port, in the exact same order that they compose the port.

The PinEvent Class

The `PinEvent` class encapsulates GPIO pin value changes. If value change events for the same GPIO pin are coalesced, the value returned is that of the last occurrence. The class consists of two constructors and one accessor.

- `PinEvent(GPIOPin pin, boolean value)`

  This constructor creates a new `PinEvent` with the specified value. The event is then time-stamped with the current time.

- `PinEvent(GPIOPin pin, boolean value, long timeStamp, int timeStampMicros)`

  This constructor creates a new `PinEvent` with the specified value and timestamp. Additional microseconds can also be added using the fourth parameter, if necessary.

- `boolean getValue()`

  This method returns a boolean indicating the GPIO pin's new value, `true` for high or `false` for low.

The PortEvent Class

The `PortEvent` class encapsulates GPIO port value changes. If value change events for the same GPIO port are coalesced, the value returned is that of the last occurrence.

- `PortEvent(GPIOPort port, boolean value)`

  This constructor creates a new `PortEvent` with the specified value. The event is then time-stamped with the current time.

- `PortEvent(GPIOPort port, boolean value, long timeStamp, int timeStampMicros)`

  This constructor creates a new `PortEvent` with the specified value and timestamp. Additional microseconds can also be added using the fourth parameter, if necessary.

- `int getValue()`

  This method returned is interpreted as an unsigned 32-bit integer representing the GPIO port's new value.