

```

1  #!/usr/bin/python
2  import re
3  import smbus2 as smbus
4
5  # =====
6  # Raspi_I2C Class
7  # =====
8
9  class Raspi_I2C(object):
10
11     @staticmethod
12     def getPiRevision():
13         "Gets the version number of the Raspberry Pi board"
14         # Revision list available at:
15         # http://elinux.org/RPi\_HardwareHistory#Board\_Revision\_History
16         try:
17             with open('/proc/cpuinfo', 'r') as infile:
18                 for line in infile:
19                     # Match a line of the form "Revision : 0002" while ignoring extra
20                     # info in front of the revision (like 1000 when the Pi was over-volted).
21                     match = re.match('Revision\s+:\s+.*(\w{4})$', line)
22                     if match and match.group(1) in ['0000', '0002', '0003']:
23                         # Return revision 1 if revision ends with 0000, 0002 or 0003.
24                         return 1
25                     elif match:
26                         # Assume revision 2 if revision ends with any other 4 chars.
27                         return 2
28                     # Couldn't find the revision, assume revision 0 like older code for
29                     # compatibility.
30                     return 0
31         except:
32             return 0
33
34     @staticmethod
35     def getPiI2CBusNumber():
36         # Gets the I2C bus number /dev/i2c#
37         return 1 if Raspi_I2C.getPiRevision() > 1 else 0
38
39     def __init__(self, address, busnum=-1, debug=False):
40         self.address = address
41         # By default, the correct I2C bus is auto-detected using /proc/cpuinfo
42         # Alternatively, you can hard-code the bus version below:
43         # self.bus = smbus.SMBus(0); # Force I2C0 (early 256MB Pi's)
44         # self.bus = smbus.SMBus(1); # Force I2C1 (512MB Pi's)
45         self.bus = smbus.SMBus(busnum if busnum >= 0 else Raspi_I2C.getPiI2CBusNumber())
46         self.debug = debug
47
48     def reverseByteOrder(self, data):
49         "Reverses the byte order of an int (16-bit) or long (32-bit) value"
50         # Courtesy Vishal Sapre
51         byteCount = len(hex(data)[2:].replace('L', '')[::-2])
52         val = 0
53         for i in range(byteCount):
54             val = (val << 8) | (data & 0xff)
55             data >>= 8
56         return val
57
58     def errMsg(self):
59         print("Error accessing 0x%02X: Check your I2C address" % self.address)
60         return -1
61
62     def write8(self, reg, value):
63         "Writes an 8-bit value to the specified register/address"
64         try:
65             self.bus.write_byte_data(self.address, reg, value)
66             if self.debug:
67                 print("I2C: Wrote 0x%02X to register 0x%02X" % (value, reg))

```

```

66     except IOError as err:
67         return self.errMsg()
68
69     def write16(self, reg, value):
70         "Writes a 16-bit value to the specified register/address pair"
71         try:
72             self.bus.write_word_data(self.address, reg, value)
73             if self.debug:
74                 print ("I2C: Wrote 0x%02X to register pair 0x%02X,0x%02X" %
75                     (value, reg, reg+1))
76         except IOError as err:
77             return self.errMsg()
78
79     def writeRaw8(self, value):
80         "Writes an 8-bit value on the bus"
81         try:
82             self.bus.write_byte(self.address, value)
83             if self.debug:
84                 print ("I2C: Wrote 0x%02X" % value)
85         except IOError as err:
86             return self.errMsg()
87
88     def writeList(self, reg, list):
89         "Writes an array of bytes using I2C format"
90         try:
91             if self.debug:
92                 print("I2C: Writing list to register 0x%02X:" % reg)
93                 print(list)
94             self.bus.write_i2c_block_data(self.address, reg, list)
95         except IOError as err:
96             return self.errMsg()
97
98     def readList(self, reg, length):
99         "Read a list of bytes from the I2C device"
100        try:
101            results = self.bus.read_i2c_block_data(self.address, reg, length)
102            if self.debug:
103                print ("I2C: Device 0x%02X returned the following from reg 0x%02X" %
104                    (self.address, reg))
105                print (results)
106            return results
107        except IOError as err:
108            return self.errMsg()
109
110    def readU8(self, reg):
111        "Read an unsigned byte from the I2C device"
112        try:
113            result = self.bus.read_byte_data(self.address, reg)
114            if self.debug:
115                print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
116                    (self.address, result & 0xFF, reg))
117            return result
118        except IOError as err:
119            return self.errMsg()
120
121    def readS8(self, reg):
122        "Reads a signed byte from the I2C device"
123        try:
124            result = self.bus.read_byte_data(self.address, reg)
125            if result > 127: result -= 256
126            if self.debug:
127                print ("I2C: Device 0x%02X returned 0x%02X from reg 0x%02X" %
128                    (self.address, result & 0xFF, reg))
129            return result
130        except IOError as err:
131            return self.errMsg()
132

```

```

133 def readU16(self, reg, little_endian=True):
134     "Reads an unsigned 16-bit value from the I2C device"
135     try:
136         result = self.bus.read_word_data(self.address, reg)
137         # Swap bytes if using big endian because read_word_data assumes little
138         # endian on ARM (little endian) systems.
139         if not little_endian:
140             result = ((result << 8) & 0xFF00) + (result >> 8)
141         if (self.debug):
142             print ("I2C: Device 0x%02X returned 0x%04X from reg 0x%02X" % (self.address,
143                                     result & 0xFFFF, reg) )
144         return result
145     except IOError as err:
146         return self.errMsg()
147
148 def readS16(self, reg, little_endian=True):
149     "Reads a signed 16-bit value from the I2C device"
150     try:
151         result = self.readU16(reg, little_endian)
152         if result > 32767: result -= 65536
153         return result
154     except IOError as err:
155         return self.errMsg()
156
157 if __name__ == '__main__':
158     try:
159         bus = Raspi_I2C(address=0)
160         print("Default I2C bus is accessible")
161     except:
162         print("Error accessing default I2C bus")

```