smbus2 Documentation

Release 0.4.2

Karl-Petter Lindegaard

Jun 05, 2022
## Contents

<table>
<thead>
<tr>
<th>Python Module Index</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>7</td>
</tr>
</tbody>
</table>
smbus2 - A drop-in replacement for smbus-cffi/smbus-python

```python
class smbus2.SMBus(bus=None, force=False)
```

```python
block_process_call(i2c_addr, register, data, force=None)
```
Executes a SMBus Block Process Call, sending a variable-size data block and receiving another variable-size response

**Parameters**

- `i2c_addr` *(int)* – i2c address
- `register` *(int)* – Register to read/write to
- `data` *(list)* – List of bytes
- `force` *(Boolean)* –

**Returns** List of bytes

**Return type** list

```python
close()
```
Close the i2c connection.

```python
enable_pec(enable=True)
```
Enable/Disable PEC (Packet Error Checking) - SMBus 1.1 and later

**Parameters**

- `enable` *(Boolean)* –

```python
i2c_rdwr(*i2c_msgs)
```
Combine a series of i2c read and write operations in a single transaction (with repeated start bits but no stop bits in between).

This method takes i2c_msg instances as input, which must be created first with `i2c_msg.read()` or `i2c_msg.write()`.

**Parameters**

- `i2c_msgs` *(i2c_msg)* – One or more i2c_msg class instances.

**Return type** None

```python
open(bus)
```
Open a given i2c bus.

**Parameters**

- `bus` *(int or str)* – i2c bus number (e.g. 0 or 1) or an absolute file path (e.g. `/dev/i2c-42`).

**Raises** `TypeError` – if type(bus) is not in (int, str)

```python
pec
```
Get and set SMBus PEC. 0 = disabled (default), 1 = enabled.

```python
process_call(i2c_addr, register, value, force=None)
```
Executes a SMBus Process Call, sending a 16-bit value and receiving a 16-bit response

**Parameters**

- `i2c_addr` *(int)* – i2c address
- `register` *(int)* – Register to read/write to
- `value` *(int)* – Word value to transmit
• force (Boolean) –
    Return type  int

**read_block_data** (*i2c_addr, register, force=None*)
Read a block of up to 32-bytes from a given register.

Parameters
• *i2c_addr* (*int*) – i2c address
• *register* (*int*) – Start register
• *force* (Boolean) –

Returns  List of bytes
Return type  list

**read_byte** (*i2c_addr, force=None*)
Read a single byte from a device.

Return type  int
Parameters
• *i2c_addr* (*int*) – i2c address
• *force* (Boolean) –

Returns  Read byte value

**read_byte_data** (*i2c_addr, register, force=None*)
Read a single byte from a designated register.

Parameters
• *i2c_addr* (*int*) – i2c address
• *register* (*int*) – Register to read
• *force* (Boolean) –

Returns  Read byte value
Return type  int

**read_i2c_block_data** (*i2c_addr, register, length, force=None*)
Read a block of byte data from a given register.

Parameters
• *i2c_addr* (*int*) – i2c address
• *register* (*int*) – Start register
• *length* (*int*) – Desired block length
• *force* (Boolean) –

Returns  List of bytes
Return type  list

**read_word_data** (*i2c_addr, register, force=None*)
Read a single word (2 bytes) from a given register.

Parameters
• *i2c_addr* (*int*) – i2c address
• register \((\text{int})\) – Register to read
• force \((\text{Boolean})\) –

Returns 2-byte word

Return type int

write_block_data \((\text{i2c_addr, register, data, force=}\text{None})\)
Write a block of byte data to a given register.

Parameters
• i2c_addr \((\text{int})\) – i2c address
• register \((\text{int})\) – Start register
• data \((\text{list})\) – List of bytes
• force \((\text{Boolean})\) –

Return type None

write_byte \((\text{i2c_addr, value, force=}\text{None})\)
Write a single byte to a device.

Parameters
• i2c_addr \((\text{int})\) – i2c address
• value \((\text{int})\) – value to write
• force \((\text{Boolean})\) –

write_byte_data \((\text{i2c_addr, register, value, force=}\text{None})\)
Write a byte to a given register.

Parameters
• i2c_addr \((\text{int})\) – i2c address
• register \((\text{int})\) – Register to write to
• value \((\text{int})\) – Byte value to transmit
• force \((\text{Boolean})\) –

Return type None

write_i2c_block_data \((\text{i2c_addr, register, data, force=}\text{None})\)
Write a block of byte data to a given register.

Parameters
• i2c_addr \((\text{int})\) – i2c address
• register \((\text{int})\) – Start register
• data \((\text{list})\) – List of bytes
• force \((\text{Boolean})\) –

Return type None

write_quick \((\text{i2c_addr, force=}\text{None})\)
Perform quick transaction. Throws IOError if unsuccessful.

: param i2c_addr: i2c address
: type i2c_addr: int
: param force: : type force: Boolean
**write_word_data**(*i2c_addr*, *register*, *value*, *force=None*)

Write a single word (2 bytes) to a given register.

Parameters

- **i2c_addr** (*int*) – i2c address
- **register** (*int*) – Register to write to
- **value** (*int*) – Word value to transmit
- **force** (*Boolean*) –

Return type None

```python
class smbus2.i2c_msg
```

As defined in i2c.h.

- **addr**
  Structure/Union member
- **buf**
  Structure/Union member
- **flags**
  Structure/Union member
- **len**
  Structure/Union member

```python
class smbus2.i2c_msg
```

```python
static read(address, length)
```

Prepares an i2c read transaction.

Parameters

- **address** – Slave address.
- **length** – Number of bytes to read.

Type address: int
Type length: int

Returns New *i2c_msg* instance for read operation.

Return type *i2c_msg*

```python
static write(address, buf)
```

Prepares an i2c write transaction.

Parameters

- **address** (*int*) – Slave address.
- **buf** (*list*) – Bytes to write. Either list of values or str.

Returns New *i2c_msg* instance for write operation.

Return type *i2c_msg*
smbus2.1
<table>
<thead>
<tr>
<th>Letter</th>
<th>Method/Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>addr</td>
<td>smbus2.i2c_msg attribute, 4</td>
</tr>
<tr>
<td>B</td>
<td>block_process_call()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td></td>
<td>buf</td>
<td>smbus2.i2c_msg attribute, 4</td>
</tr>
<tr>
<td>C</td>
<td>close()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td>E</td>
<td>enable_pec()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td>F</td>
<td>flags</td>
<td>smbus2.i2c_msg attribute, 4</td>
</tr>
<tr>
<td>I</td>
<td>i2c_msg</td>
<td>class in smbus2, 4</td>
</tr>
<tr>
<td></td>
<td>i2c_rdwr()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td>L</td>
<td>len</td>
<td>smbus2.i2c_msg attribute, 4</td>
</tr>
<tr>
<td>O</td>
<td>open()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td>P</td>
<td>pec</td>
<td>smbus2.SMBus attribute, 1</td>
</tr>
<tr>
<td></td>
<td>process_call()</td>
<td>smbus2.SMBus method, 1</td>
</tr>
<tr>
<td>R</td>
<td>read()</td>
<td>smbus2.i2c_msg static method, 4</td>
</tr>
<tr>
<td></td>
<td>read_block_data()</td>
<td>smbus2.SMBus method, 2</td>
</tr>
<tr>
<td></td>
<td>read_byte()</td>
<td>smbus2.SMBus method, 2</td>
</tr>
<tr>
<td></td>
<td>read_byte_data()</td>
<td>smbus2.SMBus method, 2</td>
</tr>
<tr>
<td></td>
<td>read_i2c_block_data()</td>
<td>smbus2.SMBus method, 2</td>
</tr>
<tr>
<td></td>
<td>read_word_data()</td>
<td>smbus2.SMBus method, 2</td>
</tr>
<tr>
<td>S</td>
<td>SMBus</td>
<td>class in smbus2, 1</td>
</tr>
<tr>
<td></td>
<td>smbus2 (module)</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>write()</td>
<td>smbus2.i2c_msg static method, 4</td>
</tr>
<tr>
<td></td>
<td>write_block_data()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
<tr>
<td></td>
<td>write_byte()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
<tr>
<td></td>
<td>write_byte_data()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
<tr>
<td></td>
<td>write_i2c_block_data()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
<tr>
<td></td>
<td>write_quick()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
<tr>
<td></td>
<td>write_word_data()</td>
<td>smbus2.SMBus method, 3</td>
</tr>
</tbody>
</table>