libpebble2 Documentation

Release 0.0.23

Pebble Technology

July 28, 2016

Contents

CHAPTER 1

Connections

Connections to the Pebble are represented by the `PebbleConnection`, connecting via a transport.

1.1 Event loops

Having connected to a Pebble, the event loop must be run for anything interesting to occur. There are three ways to do this:

1.1.1 Fire and forget

`PebbleConnection.run_async()` will spawn a new thread (called `PebbleConnection`) and run the event loop on that thread. It handles expected exceptions and will emit `log messages` at the `WARNING` level. It will also call *`PebbleConnection.fetch_watch_info()`* on your behalf before returning. This is the easiest option, and often what you want.

```
pebble.connect()
pebble.run_async()
```

1.1.2 Blocking forever

`PebbleConnection.run_sync()` will run the event loop in place. It will handle exceptions for you and emit logs at the `WARNING` level. It will not return until the watch disconnects or an error occurs.

```
pebble.connect()
pebble.run_sync()
```

1.1.3 Do it manually

Calling `PebbleConnection.pump_reader()` will (synchronously) cause exactly one message to be read from the transport, which may or may not be a message from the Pebble. It will fire all of the events for that message, and then return. It doesn't return anything.

```
pebble.connect()
while pebble.connected:
    pebble.pump_reader()
```

Note: `pump_reader` may throw exceptions on receiving malformed messages; these should probably be handled.

1.2 API

class `libpebble2.communication.`**`FirmwareVersion`**(*major*, *minor*, *patch*, *suffix*)
Represents a firmware version, in the format `major.minor.patch-suffix`.

`major`
Alias for field number 0

`minor`
Alias for field number 1

`patch`
Alias for field number 2

`suffix`
Alias for field number 3

class `libpebble2.communication.`**`PebbleConnection`**(*transport*, *log_protocol_level=None*, *log_packet_level=None*)
PebbleConnection represents the connection to a pebble; all interaction with a pebble goes through it.

Parameters

- **`transport`** (`BaseTransport`) – The underlying transport layer to communicate with the Pebble.
- **`log_packet_level`** (*int*) – If not None, the log level at which to log decoded messages sent and received.
- **`log_protocol_level`** (*int*) – int If not None, the log level at which to log raw messages sent and received.

`connect`()
Synchronously initialises a connection to the Pebble. Once it returns, a valid connection will be open.

`connected`

Returns `True` if currently connected to a Pebble; otherwise `False`.

`fetch_watch_info`()
This method should be called before accessing *watch_info*, *firmware_version* or *watch_platform*. Blocks until it has fetched the required information.

`firmware_version`
Provides information on the connected Pebble, including its firmware version, language, capabilities, etc.

Return type *WatchVersionResponse*

`get_endpoint_queue`(*endpoint*)
Returns a *BaseEventQueue* from which messages to the given `endpoint` can be read.

This is useful if you need to make sure that you receive all messages to an endpoint, without risking dropping some due to time in between *read_from_endpoint()* calls.

Parameters **`endpoint`** (*PacketType*) – The endpoint to read from

Returns

pump_reader()
Synchronously reads one message from the watch, blocking until a message is available. All events caused by the message read will be processed before this method returns.

Note: You usually don't need to invoke this method manually; instead, see *run_sync()* and *run_async()*.

---

read_from_endpoint(*endpoint*, *timeout=15*)
Blocking read from an endpoint. Will block until a message is received, or it times out. Also see *get_endpoint_queue()* if you are considering calling this in a loop.

Warning: Avoid calling this method from an endpoint callback; doing so is likely to lead to deadlock.

Note: If you're reading a response to a message you just sent, *send_and_read()* might be more appropriate.

Parameters

- **endpoint** (*PacketType*) – The endpoint to read from.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The message read from the endpoint; of the same type as passed to `endpoint`.

read_transport_message(*origin*, *message_type*, *timeout=15*)
Blocking read of a transport message that does not indicate a message from the Pebble. Will block until a message is received, or it times out.

Warning: Avoid calling this method from an endpoint callback; doing so is likely to lead to deadlock.

Parameters

- **origin** – The type of *MessageTarget* that triggers the message.
- **message_type** – The class of the message to read from the transport.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The object read from the transport; of the same type as passed to `message_type`.

register_endpoint(*endpoint*, *handler*)
Register a handler for a message received from the Pebble.

Parameters

- **endpoint** (*PacketType*) – The type of *PebblePacket* that is being listened for.
- **handler** (*callable*) – A callback to be called when a message is received.

Returns A handle that can be passed to *unregister_endpoint()* to remove the handler.

register_raw_inbound_handler(*handler*)
Register a handler for all outgoing messages received from the Pebble. Transport framing is not included. In most cases you should not need to use this; consider using *register_endpoint()* instead.

Parameters **handler** (*callable*) – A callback to be called when any message is received.

Returns A handle that can be passed to *unregister_endpoint()* to remove the handler.

register_raw_outbound_handler(*handler*)
Register a handler for all outgoing messages to be sent to the Pebble. Transport framing is not included.

Parameters **handler** (*callable*) – A callback to be called when any message is received.

Returns A handle that can be passed to *unregister_endpoint()* to remove the handler.

register_transport_endpoint(*origin*, *message_type*, *handler*)
Register a handler for a message received from a transport that does not indicate a message from the connected Pebble.

Parameters

- **origin** – The type of *MessageTarget* that triggers the message
- **message_type** – The class of the message that is expected.
- **handler** (*callable*) – A callback to be called when a message is received.

Returns A handle that can be passed to *unregister_endpoint()* to remove the handler.

run_async()
Spawns a new thread that runs the message loop until the Pebble disconnects. `run_async` will call *fetch_watch_info()* on your behalf, and block until it receives a response.

run_sync()
Runs the message loop until the Pebble disconnects. This method will block until the watch disconnects or a fatal error occurs.

For alternatives that don't block forever, see *pump_reader()* and *run_async()*.

send_and_read(*packet*, *endpoint*, *timeout=15*)
Sends a packet, then returns the next response received from that endpoint. This method sets up a listener before it actually sends the message, avoiding a potential race.

Warning: Avoid calling this method from an endpoint callback; doing so is likely to lead to deadlock.

Parameters

- **packet** (PebblePacket) – The message to send.
- **endpoint** (*PacketType*) – The endpoint to read from
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The message read from the endpoint; of the same type as passed to `endpoint`.

send_packet(*packet*)
Sends a message to the Pebble.

Parameters **packet** (PebblePacket) – The message to send.

send_raw(*message*)
Sends a raw binary message to the Pebble. No processing will be applied, but any transport framing should be omitted.

Parameters **message** (*bytes*) – The message to send to the pebble.

unregister_endpoint(*handle*)
Removes a handler registered by *register_transport_endpoint()*, *register_endpoint()*, *register_raw_outbound_handler()* or *register_raw_inbound_handler()*.

Parameters **handle** – A handle returned by the register call to be undone.

watch_info
Returns information on the connected Pebble, including its firmware version, language, capabilities, etc.

Return type *WatchVersionResponse*

watch_model

Returns The model of the watch.

Return type *Model*

watch_platform
A string naming the platform of the watch ('aplite', 'basalt', 'chalk', or 'unknown').

Return type str

CHAPTER 2

Transports

A transport represents a channel over which libpebble2 can communicate with a Pebble. Transports can also support sending and receiving messages not destined for the Pebble — for instance, the WebSocket transport can install a JavaScript app in the phone app. Two transports are currently provided, but it would be easy to add more.

Transports are usually only accessed directly by *PebbleConnection*, but it can be useful to use them directly to interact with the transport instead of a Pebble. The transport can be accessed using the *transport* attribute of the *PebbleConnection*.

The origin or destination of a message is indicated using a "message target". Messages to or from the watch use `MessageTargetWatch`; other transports may define additional targets of their own, which they can use to route messages elsewhere.

2.1 BaseTransport

BaseTransport defines the functionality expected of any transport. All transports should inherit from *BaseTransport*.

class `libpebble2.communication.transports.`**BaseTransport**

connect()
Synchronously connect to the Pebble. Once this method returns, libpebble2 should be able to safely send messages to the connected Pebble.

Ordinarily, this method should only be called by *PebbleConnection*.

connected

Returns `True` if the transport is currently connected; otherwise `False`.

must_initialise

Returns `True` if libpebble2 is responsible for negotiating the connection; otherwise `False`.

read_packet()
Synchronously read a message. This message could be from the Pebble(in which case it will be a `PebblePacket`), or it could be from the transport, in which case the result is transport-defined. The origin of the result is indicated by the returned `MessageTarget`.

Returns (`MessageTarget`, *libpebble2.protocol.base.PebblePacket*)

send_packet(*message*, *target=MessageTargetWatch()*)
Send a message. This message could be to the Pebble (in which case it must be a `PebblePacket`), or to the transport (in which case the message type is transport-defined).

Parameters

- **message** (PebblePacket) – Message to send.
- **target** (MessageTarget) – Target for the message

class libpebble2.communication.transports.**MessageTargetWatch**
Bases: *libpebble2.communication.transports.MessageTarget*

class libpebble2.communication.transports.**MessageTarget**

2.2 WebSocket transport

The WebSocket transport connects to a phone running the Pebble mobile app using the "Developer Connection", which exposes a WebSocket server on the phone. By default it runs on port 9000.

```
>>> pebble = PebbleConnection(WebsocketTransport("ws://192.168.204:9000/"))
```

class libpebble2.communication.transports.websocket.**WebsocketTransport**(*url*)
Bases: *libpebble2.communication.transports.BaseTransport*

Represents a connection via WebSocket to a phone running the Pebble mobile app, which is in turn connected to a Pebble over Bluetooth.

Parameters **url** – The WebSocket URL to connect to, in standard format (e.g. `ws://localhost:9000/`)

class libpebble2.communication.transports.websocket.**MessageTargetPhone**
Bases: *libpebble2.communication.transports.MessageTarget*

Indicates that the message is directed at a connected phone running the Pebble mobile app. For this purpose, pypkjs counts as a phone.

2.3 QEMU transport

The QEMU transport connects to an instance of Pebble QEMU via Pebble QEMU Protocol. Note that, due to how QEMU is implemented, the watch will not necessarily notice connections or disconnections over this transport.

Messages directed at the emulator itself, rather than the firmware running on it, can be sent using *MessageTargetQemu*.

```
>>> pebble = PebbleConnection(QemuTransport("localhost", 12344))
```

class libpebble2.communication.transports.qemu.**QemuTransport**(*host='127.0.0.1'*, *port=12344*)
Bases: *libpebble2.communication.transports.BaseTransport*

Represents a connection to a Pebble QEMU instance.

Parameters

- **host** (*str*) – The host on which the QEMU instance is running.
- **port** (*int*) – The port on which the QEMU instance has exposed its Pebble QEMU Protocol port.

BUFFER_SIZE = 2048
Number of bytes read from the socket at a time.

class libpebble2.communication.transports.qemu.**MessageTargetQemu**(*protocol=None*, *raw=False*)

Bases: *libpebble2.communication.transports.MessageTarget*

Indicates that a message is directed at QEMU, rather than the firmware running on it. If `raw` is `True`, the message should be a binary message (without framing), with QEMU protocol indicated by `protocol`. Otherwise, the message should be a *PebblePacket* from `protocol`.

Parameters

- **protocol** (int) – The protocol to send to, if sending a `raw` message
- **raw** (bool) – If `True`, the message is pre-serialised and will be sent as-is after adding framing.

2.4 Serial transport

It is possible to connect directly to the Pebble using *SerialTransport*. This transport uses the operating system's built-in Bluetooth serial support to communicate with the watch using pyserial. Using this transport requires the Pebble to already be paired with the computer. Recall that the Pebble may only connect to one device at a time; disconnect any connected phones (e.g. by disabling Bluetooth) before attempting to pair with your computer or use this transport.

Since this transport connects directly to the watch, it does not define any other message targets.

class libpebble2.communication.transports.serial.**SerialTransport**(*device*)

Bases: *libpebble2.communication.transports.BaseTransport*

Represents a direct connection to a physical Pebble paired to the computer via Bluetooth serial. This transport expects to be given a device file over which it can communicate with the watch via Bluetooth.

> **Warning:** Using this transport may cause occasional kernel panics on some versions of OS X.

Parameters **device** (*str*) – The path to the device file (on OS X, often of the form `/dev/cu.PebbleTimeXXXX-SerialPo` or `/dev/cu.PebbleXXXX-SerialPortSe`).

CHAPTER 3

Protocol handling

libpebble2 provides a simple DSL for defining Pebble Protocol messages, accounting for various quirks in the Pebble Protocol, such as the four different ways of defining strings and mixed endianness.

3.1 Defining messages

All messages inherit from *PebblePacket*, which uses metaclass magic (from `PacketType`) to parse the definitions. An empty message would look like this:

```
class SampleMessage(PebblePacket):
    pass
```

This message is not very interesting — it represents a zero-length, unidentifiable packet. Despite this, it can be useful in conjunction with certain field types, such as `Union`.

3.1.1 Metadata

To add some useful information about our message, we can define a `Meta` inner class inside it:

```
class SampleMessage(PebblePacket):
    class Meta:
        endpoint = 0xbead
        endianness = '<'
```

This defines our `SampleMessage` as being a little-endian Pebble Protocol message that should be sent to endpoint 0xbead.

The following attributes on `Meta` are meaningful (but all are optional):

- `endpoint` — defines the Pebble Protocol endpoint to which the message should be sent.
- `endianness` — defines the endianness of the message. Use `'<'` for little-endian or `'>'` for big-endian.
- `register` — if specified and `False`, the message will not be registered for parsing when received, even if `endpoint` is specified. This can be useful if the protocol design is asymmetric and ambiguous.

Note: `Meta` is not inherited if you subclass a `PebblePacket`. In particular, you will probably want to re-specify `endianness` when doing this. The default endianness is **big-endian**.

We can now use this class to send an empty message to the watch, or receive one back!

```
>>> pebble.send_packet(SampleMessage())
```

3.1.2 Fields

Empty messages are rarely useful. To actually send some information, we can add more attributes to our messages. For instance, let's say we want to specify a time message that looks like this:

Offset	Length	Type	Value
0	4	uint32_t	Seconds since 1970 (unix time, UTC)
4	2	uint16_t	UTC offset in minutes, including DST
6	1	uint8_t	Length of the timezone region name
7	...	char *	The timezone region name

We could represent that packet like this:

```
class SetUTC(PebblePacket):
    unix_time = Uint32()
    utc_offset_mins = Int16()
    tz_name = PascalString()
```

The lengths and offsets are determined automatically. Also notice that we didn't have to include the length explicitly — including a length byte before a string is a sufficiently common pattern that it has a dedicated `PascalString` field. This definition works:

```
>>> from binascii import hexlify
>>> message = SetUTC(unix_time=1436165495, utc_offset_mins=-420, tz_name=u"America/Los_Angeles")
>>> hexlify(message.serialise())
'559a2577fe5c13416d65726963612f4c6f735f416e67656c6573'
>>> SetUTC.parse('559a2577fe5c13416d65726963612f4c6f735f416e67656c6573'.decode('hex'))
(SetUTC(unix_time=1436165495, utc_offset=-420, tz_name=America/Los_Angeles), 26)
```

(*`parse()`* returns a `(message, consumed_bytes)` tuple.)

Which is nice, but isn't usable as a Pebble Protocol message — after all, we don't have an endpoint. It also turns out that this isn't *actually* a message you can send to the Pebble; rather, it's merely one of four possible messages to the "Time" endpoint. How can we handle that? With a `Union`! Let's build the whole Time message:

```
class GetTimeRequest(PebblePacket):
    pass

class GetTimeResponse(PebblePacket):
    localtime = Uint32()

class SetLocaltime(PebblePacket):
    localtime = Uint32()

class SetUTC(PebblePacket):
    unix_time = Uint32()
    utc_offset_mins = Int16()
    tz_name = PascalString()

class TimeMessage(PebblePacket):
    class Meta:
        endpoint = 0xb
        endianness = '>'  # big endian

    command = Uint8()
    message = Union(command, {
```

```
        0x00: GetTimeRequest,
        0x01: GetTimeResponse,
        0x02: SetLocaltime,
        0x03: SetUTC,
    })
```

`TimeMessage` is now our Pebble Protocol message. Its `Meta` class contains two pieces of information; the endpoint and the endianness of the message (which is actually the default). It consists of two fields: a `command`, which is just a `uint8_t`, and a `message`. Union applies the endianness specified in `TimeMessage` to the other classes it references.

During deserialisation, the `Union` will use the value of `command` to figure out which member of the union to use, then use that class to parse the remainder of the message. During serialisation, `Union` will inspect the type of the provided `message`:

```
>>> message = TimeMessage(message=SetUTC(unix_time=1436165495, utc_offset_mins=-420, tz_name=u"Americ
#  We don't have to set command because Union does that for us.
>>> hexlify(message.serialise_packet())
'001b000b03559a2577fe5c13416d65726963612f4c6f735f416e67656c6573'
>>> PebblePacket.parse_message('001b000b03559a2577fe5c13416d65726963612f4c6f735f416e67656c6573'.decod
(TimeMessage(kind=3, message=SetUTC(unix_time=1436165495, utc_offset=-420, tz_name=America/Los_Angele
>>> pebble.send_packet(message)
```

And there we go! We encoded a pebble packet, then asked the general `PebblePacket` to deserialise it for us. But wait: how did `PebblePacket` know to return a `TimeMessage`?

When defining a subclass of `PebblePacket`, it will automatically be registered in an internal "packet registry" if it has an `endpoint` specified. Sometimes this behaviour is undesirable; in this case, you can specify `register = False` to disable this behaviour.

3.2 API

3.2.1 Packets

class `libpebble2.protocol.base.`**`PebblePacket`**`(`***kwargs*`)`

Represents some sort of Pebble Protocol message.

A PebblePacket can have an inner class named `Meta` containing some information about the property:

endpoint	The Pebble Protocol endpoint that is represented by this message.
endianness	The endianness of the packet. The default endianness is big-endian, but it can be overridden by packets and fields, with the priority:
register	If set to `False`, the packet will not be registered and thus will be ignored by `parse_message()`. This is useful when messages are ambiguous, and distinguished only by whether they are sent to or from the Pebble.

A sample packet might look like this:

```
class AppFetchResponse(PebblePacket):
    class Meta:
        endpoint = 0x1771
        endianness = '<'
        register = False
```

```
        command = Uint8(default=0x01)
        response = Uint8(enum=AppFetchStatus)
```

Parameters ****kwargs** – Initial values for any properties on the object.

classmethod parse(*message*, *default_endianness='!'*)
Parses a message without any framing, returning the decoded result and length of message consumed. The result will always be of the same class as `parse()` was called on. If the message is invalid, *PacketDecodeError* will be raised.

Parameters

- **message** (*bytes*) – The message to decode.
- **default_endianness** – The default endianness, unless overridden by the fields or class metadata. Should usually be left at `None`. Otherwise, use `'<'` for little endian and `'>'` for big endian.

Returns `(decoded_message, decoded length)`

Return type (`PebblePacket`, `int`)

classmethod parse_message(*message*)
Parses a message received from the Pebble. Uses Pebble Protocol framing to figure out what sort of packet it is. If the packet is registered (has been defined and imported), returns the deserialised packet, which will not necessarily be the same class as this. Otherwise returns `None`.

Also returns the length of the message consumed during deserialisation.

Parameters **message** (*bytes*) – A serialised message received from the Pebble.

Returns `(decoded_message, decoded length)`

Return type (`PebblePacket`, `int`)

serialise(*default_endianness=None*)
Serialise a message, without including any framing.

Parameters **default_endianness** (*str*) – The default endianness, unless overridden by the fields or class metadata. Should usually be left at `None`. Otherwise, use `'<'` for little endian and `'>'` for big endian.

Returns The serialised message.

Return type bytes

serialise_packet()
Serialise a message, including framing information inferred from the `Meta` inner class of the packet. `self.Meta.endpoint` must be defined to call this method.

Returns A serialised message, ready to be sent to the Pebble.

3.2.2 Field types

Padding	Represents some unused bytes.
Boolean	Represents a `bool`.
Uint8	Represents a `uint8_t`.
Uint16	Represents a `uint16_t`.

Table 3.1 – continued from previous page

Uint32	Represents a `uint32_t`.
Uint64	Represents a `uint64_t`.
Int8	Represents an `int8_t`.
Int16	Represents an `int16_t`.
Int32	Represents an `int32_t`.
Int64	Represents an `int64_t`.
FixedString	Represents a "fixed-length" string.
NullTerminatedString	Represents a null-terminated, UTF-8 encoded string (i.e.
PascalString	Represents a UTF-8-encoded string that is prefixed with a length byte.
FixedList	Represents a list of either `PebblePacket`s or `Field`s with either a fixed number of entries, a fixe
PascalList	Represents a list of *PebblePacket*s, each of which is prefixed with a byte indicating its length.
Union	Represents a union of some other set of fields or packets, determined by some other field (`determi`
Embed	Embeds another *PebblePacket*.

class `libpebble2.protocol.base.types.`**`Field`**(*default=None*, *endianness=None*, *enum=None*)

Base class for Pebble Protocol fields. This class does nothing; only subclasses are useful.

Parameters

- **`default`** – The default value of the field, if nothing else is specified.
- **`endianness`** (*str*) – The endianness of the field. By default, inherits from packet, or its parent packet, etc. Use `"<"` for little endian or `">"` for big endian.
- **`enum`** (*Enum*) – An `Enum` that represents the possible values of the field.

`buffer_to_value`(*obj*, *buffer*, *offset*, *default_endianness='!'*)

Converts the bytes in `buffer` at `offset` to a native Python value. Returns that value and the number of bytes consumed to create it.

Parameters

- **`obj`** (`PebblePacket`) – The parent *PebblePacket* of this field
- **`buffer`** (*bytes*) – The buffer from which to extract a value.
- **`offset`** (*int*) – The offset in the buffer to start at.
- **`default_endianness`** (*str*) – The default endianness of the value. Used if `endianness` was not passed to the *Field* constructor.

Returns (value, length)

Return type (`object`, `int`)

`struct_format` = None

A format code for use in `struct.pack()`, if using the default implementation of *buffer_to_value()* and *value_to_bytes()*

`value_to_bytes`(*obj*, *value*, *default_endianness='!'*)

Converts the given value to an appropriately encoded string of bytes that represents it.

Parameters

- **`obj`** (`PebblePacket`) – The parent *PebblePacket* of this field
- **`value`** – The python value to serialise.
- **`default_endianness`** (*str*) – The default endianness of the value. Used if `endianness` was not passed to the *Field* constructor.

Returns The serialised value

Return type bytes

class libpebble2.protocol.base.types.**Int8**(*default=None*, *endianness=None*, *enum=None*)
Represents an int8_t.

class libpebble2.protocol.base.types.**Uint8**(*default=None*, *endianness=None*, *enum=None*)
Represents a uint8_t.

class libpebble2.protocol.base.types.**Int16**(*default=None*, *endianness=None*, *enum=None*)
Represents an int16_t.

class libpebble2.protocol.base.types.**Uint16**(*default=None*, *endianness=None*, *enum=None*)
Represents a uint16_t.

class libpebble2.protocol.base.types.**Int32**(*default=None*, *endianness=None*, *enum=None*)
Represents an int32_t.

class libpebble2.protocol.base.types.**Uint32**(*default=None*, *endianness=None*, *enum=None*)
Represents a uint32_t.

class libpebble2.protocol.base.types.**Int64**(*default=None*, *endianness=None*, *enum=None*)
Represents an int64_t.

class libpebble2.protocol.base.types.**Uint64**(*default=None*, *endianness=None*, *enum=None*)
Represents a uint64_t.

class libpebble2.protocol.base.types.**Boolean**(*default=None*, *endianness=None*, *enum=None*)
Represents a bool.

class libpebble2.protocol.base.types.**UUID**(*default=None*, *endianness=None*, *enum=None*)
Represents a UUID, represented as a 16-byte array (uint8_t[16]). The Python representation is a UUID. Endianness is ignored.

class libpebble2.protocol.base.types.**Union**(*determinant*, *contents*, *accept_missing=False*, *length=None*)
Represents a union of some other set of fields or packets, determined by some other field (determinant).

Example usage:

```
command = Uint8()
data = Union(command, {
    0: SomePacket,
    1: SomeOtherPacket,
    2: AnotherPacket
})
```

Parameters

- **determinant** (Field) – The field that is used to determine which possible entry to use.
- **contents** (*dict*) – A dict mapping values of determinant to either *Field*s or PebblePackets that this *Union* can represent. This dictionary is inverted for use in serialisation, so it should be a one-to-one mapping.
- **accept_missing** (*bool*) – If True, the *Union* will tolerate receiving unknown values, considering them to be None.
- **length** (*int*) – An optional *Field* that should contain the length of the *Union*. If provided, the field will be filled in on serialisation, and taken as a *maximum* length during deserialisation.

class libpebble2.protocol.base.types.**Embed**(*packet*, *length=None*)
Embeds another *PebblePacket*. Useful for implementing repetitive packets.

Parameters **packet** (PebblePacket) – The packet to embed.

class libpebble2.protocol.base.types.**Padding**(*length*)
Represents some unused bytes. During deserialisation, length bytes are skipped; during serialisation, length 0x00 bytes are added.

Parameters **length** (*int*) – The number of bytes of padding.

class libpebble2.protocol.base.types.**PascalString**(*null_terminated=False*, *count_null_terminator=True*, **args*, ***kwargs*)
Represents a UTF-8-encoded string that is prefixed with a length byte.

Parameters

- **null_terminated** (*bool*) – If True, a zero byte is appended to the string and included in the length during serialisation. The string is always terminated at the first zero byte during deserialisation, regardless of the value of this argument.
- **count_null_terminator** (*bool*) – If True, any appended zero byte is not counted in the length of the string. This actually comes up.

class libpebble2.protocol.base.types.**NullTerminatedString**(*default=None*, *endianness=None*, *enum=None*)
Represents a null-terminated, UTF-8 encoded string (i.e. a C string).

class libpebble2.protocol.base.types.**FixedString**(*length=None*, ***kwargs*)
Represents a "fixed-length" string. "Fixed-length" here has one of three possible meanings:

•The length is determined by another *Field* in the PebblePacket. For this effect, pass in a *Field* for length. To deserialise correctly, this field *must* appear before the *FixedString*.

•The length is fixed by the protocol. For this effect, pass in an int for length.

•The string uses the entire remainder of the packet. For this effect, omit length (or pass None).

Parameters **length** (*Field* | int) – The length of the string.

class libpebble2.protocol.base.types.**PascalList**(*member_type*, *count=None*)
Represents a list of *PebblePacket*s, each of which is prefixed with a byte indicating its length.

Parameters

- **member_type** (*type*) – The type of *PebblePacket* in the list.
- **count** (Field) – If specified, the a *Field* that contains the number of entries in the list. On serialisation, the count is filled in with the number of entries. On deserialisation, it is interpreted as a maximum; it is not an error for the packet to end prematurely.

class libpebble2.protocol.base.types.**FixedList**(*member_type*, *count=None*, *length=None*)
Represents a list of either PebblePackets or *Field*s with either a fixed number of entries, a fixed length (in bytes), or both. There are no dividers between entries; the members must be fixed-length.

If neither count nor length is set, members will be read until the end of the buffer.

Parameters

- **member_type** – Either a *Field* instance or a PebblePacket subclass that represents the members of the list.

- **count** – A *Field* containing the number of elements in the list. On serialisation, will be set to the number of members. On deserialisation, is treated as a maximum.
- **length** – A *Field* containing the number of bytes in the list. On serialisation, will be set to the length of the serialised list. On deserialisation, is treated as a maximum.

class libpebble2.protocol.base.types.**BinaryArray**(*length=None*, ***kwargs*)

An array of arbitrary bytes, represented as a Python bytes object. The length can be either a *Field*, an int, or omitted.

Parameters **length** (*Field | int*) – The length of the array:

- If it's a *Field*, the value of that field is read during deserialisation and written during seiralisation.
- If it's an int, that many bytes are always read.
- If it is None, bytes are read until the end of the message.

class libpebble2.protocol.base.types.**Optional**(*actual_field*, ***kwargs*)

Represents an optional field. It is usually an error during deserialisation for fields to be omitted. If that field is *Optional*, it will be left at its default value and ignored.

Parameters **actual_field** (Field) – The field that is being made optional.

CHAPTER 4

Services

libpebble2 provides the following services:

4.1 AppMessage

The AppMessage service is primarily used for interacting with apps via the Pebble AppMessage protocol. AppMessage represents messages as flat dictionaries, with integer keys and arbitrary values. Because AppMessage dictionary values express more type information than Python types can, wrappers are provided for the relevant types.

class libpebble2.services.appmessage.**AppMessageService**(*pebble*, *message_type=AppMessage*)

Provides a mechanism for sending and receiving AppMessages to and from the Pebble.

Incoming messages will trigger an `appmessage` event with the arguments `(transaction_id, app_uuid, data)`, where `data` is a python `dict()` containing the received values as native Python types.

AppMessageService can also be used to interact with non-AppMessage endpoints that use the same protocol, such as the legacy app state endpoint.

Parameters

- **pebble** (PebbleConnection) – The connection on which to operate.
- **message_type** (PebblePacket) – The endpoint to operate on, if not the default `AppMessage` endpoint.

send_message(*target_app*, *dictionary*)

Send a message to the given app, which should be currently running on the Pebble (unless using a non-standard AppMessage endpoint, in which case its rules apply).

AppMessage can only represent flat dictionaries with integer keys; as such, `dictionary` must be flat and have integer keys.

Because the AppMessage dictionary type is more expressive than Python's native types allow, all entries in the dictionary provided must be wrapped in one of the value types:

AppMessageService type	C type	Python type
Uint8	uint8_t	int
Uint16	uint16_t	int
Uint32	uint32_t	int
Int8	int8_t	int
Int16	int16_t	int
Int32	int32_t	int
CString	char *	str
ByteArray	uint8_t *	bytes

For instance:

```
appmessage.send_message(UUID("6FEAF2DE-24FA-4ED3-AF66-C853FA6E9C3C"), {
    16: Uint8(62),
    6428356: CString("friendship"),
})
```

Parameters

- **target_app** (*UUID*) – The UUID of the app to which to send a message.
- **dictionary** (*dict*) – The dictionary to send.

Returns The transaction ID sent message, as used in the ack and nack events.

Return type int

shutdown()
Unregisters the *AppMessageService* from the PebbleConnection that was passed into the constructor. After calling this method, no more events will be fired.

class libpebble2.services.appmessage.**Uint8**(*value*)
Represents a uint8_t

class libpebble2.services.appmessage.**Uint16**(*value*)
Represents a uint16_t

class libpebble2.services.appmessage.**Uint32**(*value*)
Represents a uint32_t

class libpebble2.services.appmessage.**Int8**(*value*)
Represents an int8_t

class libpebble2.services.appmessage.**Int16**(*value*)
Represents an int16_t

class libpebble2.services.appmessage.**Int32**(*value*)
Represents an int32_t

class libpebble2.services.appmessage.**CString**(*value*)
Represents a char *

class libpebble2.services.appmessage.**ByteArray**(*value*)
Represents a uint8_t *

4.2 BlobDB

The BlobDB service provides a mechanism for interacting with the Pebble BlobDB service. The service handles multiple messages in flight, retries, and provides callbacks for completion and failure. A *SyncWrapper* is provided

that can be passed any blobdb method and will block until it completes.

class libpebble2.services.blobdb.**BlobDBClient**(*pebble*, *timeout=5*)
Provides a mechanism for interacting with the Pebble's BlobDB service. All methods are asynchronous. Messages will be retried automatically if they time out, but all error responses from the watch are considered final and will be reported.

If you want to interact synchronously with BlobDB, see *SyncWrapper*.

Parameters

- **pebble** (PebbleConnection) – The pebble to connect to.
- **timeout** (*int*) – The timeout before resending a BlobDB command.

clear(*database*, *callback=None*)
Wipe the given database. This only affects items inserted remotely; items inserted on the watch (e.g. alarm clock timeline pins) are not removed.

Parameters

- **database** (BlobDatabaseID) – The database to wipe.
- **callback** – A callback to be called on success or failure.

delete(*database*, *key*, *callback=None*)
Delete an item from the given database.

Parameters

- **database** (BlobDatabaseID) – The database from which to delete the value.
- **key** (*uuid.UUID*) – The key to delete.
- **callback** – A callback to be called on success or failure.

insert(*database*, *key*, *value*, *callback=None*)
Insert an item into the given database.

Parameters

- **database** (BlobDatabaseID) – The database into which to insert the value.
- **key** (*uuid.UUID*) – The key to insert.
- **value** (*bytes*) – The value to insert.
- **callback** – A callback to be called on success or failure.

class libpebble2.services.blobdb.**SyncWrapper**(*method*, **args*, ***kwargs*)
Wraps a *BlobDBClient* call and returns when it completes.

Use it like this:

```
SyncWrapper(blobdb_client.insert, some_key, some_value).wait()
```

Parameters

- **method** – The method to call.
- **args** – Arguments to pass to the method.

4.3 GetBytes

class libpebble2.services.getbytes.**GetBytesService**(*pebble*)
Synchronously retrieves data from the watch over GetBytes.

Parameters **pebble** (PebbleConnection) – The Pebble to send data to.

get_coredump(*require_fresh=False*)
Retrieves a coredump, if one exists. Raises *GetBytesError* on failure.

Parameters **require_fresh** (*bool*) – If true, coredumps that have already been read are considered to not exist.

Returns The retrieved coredump

Return type bytes

get_file(*filename*)
Retrieves a PFS file from the watch. This only works on watches running non-release firmware. Raises *GetBytesError* on failure.

Returns The retrieved file

Return type bytes

get_flash_region(*offset*, *length*)
Retrieves the contents of a region of flash from the watch. This only works on watches running non-release firmware. Raises *GetBytesError* on failure.

Returns The retrieved data

Return type bytes

register_handler(*event*, *handler*)
Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

4.4 App Installation

class libpebble2.services.install.**AppInstaller**(*pebble*, *pbw_path*, *blobdb_client=None*)
Installs an app on the Pebble via Pebble Protocol.

Parameters

- **pebble** (PebbleConnection) – The PebbleConnection over which to install the app.
- **pbw_path** (*str*) – The path to the PBW file to be installed on the filesystem.
- **blobdb_client** (BlobDBClient) – An optional BlobDBClient to use, if one already exists. If omitted, one will be created.

install()
Installs an app. Blocks until the installation is complete, or raises AppInstallError if it fails.

While this method runs, "progress" events will be emitted regularly with the following signature:

```
(sent_this_interval, sent_total, total_size)
```

register_handler(*event*, *handler*)
Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

total_sent = None
Total number of bytes sent so far.

total_size = None
Total number of bytes to send.

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

4.5 Notifications

This sends simple notifications to the watch. It could probably use some improvement.

class libpebble2.services.notifications.**Notifications**(*pebble*, *blobdb=None*)
Sends notifications.

Parameters

- **pebble** (PebbleConnection) – The Pebble to send a notification to.
- **blobdb** – An existing BlobDBClient, if any. If necessary, one will be created.

send_notification(*subject=''*, *message=''*, *sender=''*, *source=None*, *actions=None*)
Sends a notification. Blocks as long as necessary.

Parameters

- **subject** (*str*) – The subject.
- **message** (*str*) – The message.
- **sender** (*str*) – The sender.
- **source** (LegacyNotification.Source) – The source of the notification

:param actions Actions to be sent with a notification (list of TimelineAction objects) :type actions list

4.6 Putbytes

class libpebble2.services.putbytes.**PutBytes**(*pebble*, *object_type*, *object*, *bank=None*, *filename=''*, *app_install_id=None*)
Synchronously sends data to the watch over PutBytes.

Parameters

- **pebble** (PebbleConnection) – The Pebble to send data to.
- **object_type** (*PutBytesType*) – The type of data being sent.
- **object** (*bytes*) – The data to send.
- **bank** (*int*) – The bank to install the data to, if applicable.
- **filename** (*str*) – The filename of the data, if applicable
- **app_install_id** (*int*) – This is used during app installations on 3.x. It is mutually exclusive with bank and filename.

register_handler(*event*, *handler*)
Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

send()
Sends the object to the watch. Block until completion, or raises *PutBytesError* on failure.

During transmission, a "progress" event will be periodically emitted with the following signature:

```
(sent_this_interval, sent_so_far, total_object_size)
```

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

4.7 Screenshots

This service takes screenshots and returns them in 8-bit ARGB format. The resulting images are lists of bytearrays, but can easily be converted to PNGs using pypng:

```
import png
image = Screenshot(pebble).grab_image()
png.from_array(image).save('screenshot.png')
```

class libpebble2.services.screenshot.**Screenshot**(*pebble*)
Takes a screenshot from the watch.

Parameters **pebble** (PebbleConnection) – The pebble of which to take a screenshot.

grab_image()
Takes a screenshot. Blocks until completion, or raises a *ScreenshotError* on failure.

While this method is executing, "progress" events will periodically be emitted with the following signature:

```
(downloaded_so_far, total_size)
```

Returns A list of bytearrays in RGB8 format, where each bytearray is one row of the image.

register_handler(*event*, *handler*)
Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

4.8 Voice

This service handles voice control endpoint messages, parses the data and exposes it for external tools. It also allows voice control messages to be sent to a Pebble. It does not implement the state machine for ordering voice control messages correctly: this must be handled by the user of the service.

4.8.1 Events

The service exposes the following events, which can be subscribed to with VoiceServer.register_handler:

- `session_setup` - Session setup request received
- `audio_frame` - Audio data frame received
- `audio_stop` - Audio data stopped

4.8.2 Voice Protocol Sequencing

The correct sequencing for communicating with the Pebble smartwatch is as follows:

Pebble-terminated sessions:

This is the normal sequence of communication. The Server should wait until it receives a stop message from the Pebble before sending the dictation result.

Message	Sender	Event/Function
Session setup request	Pebble	`session_setup`
Session setup result	Server	*`VoiceService.send_session_setup_result`*
Audio data (n frames)	Pebble	`audio_frame`
Audio stop	Pebble	`audio_stop`
Dictation result	Server	*`VoiceService.send_dictation_result`*

Server-terminated sessions:

If an error occurs a server can terminate the session by sending an audio stop message followed by the dictation result. The dictation result should always be sent.

Message	Sender	Event/Function
Session setup request	Pebble	`session_setup`
Session setup result	Server	*`VoiceService.send_session_setup_result`*
Audio data (n frames)	Pebble	`audio_frame`
Audio stop	Server	*`VoiceService.send_stop_audio`*
Dictation result	Server	*`VoiceService.send_dictation_result`*

class `libpebble2.services.voice.`**`VoiceService`**(*pebble*)

Service to expose voice control to external tools

Parameters **`pebble`** (`PebbleConnection`) – The pebble with which to establish a voice session.

`SESSION_ID_INVALID` = **0**

`register_handler`(*event*, *handler*)

Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

send_dictation_result(*result*, *sentences=None*, *app_uuid=None*)
Send the result of a dictation session

Parameters

- **result** (*DictationResult*) – Result of the session
- **sentences** – list of sentences, each of which is a list of words and punctuation
- **app_uuid** (*uuid.UUID*) – UUID of app that initiated the session

send_session_setup_result(*result*, *app_uuid=None*)
Send the result of setting up a dictation session requested by the watch

Parameters

- **result** (SetupResult) – result of setting up the session
- **app_uuid** (*uuid.UUID*) – UUID of app that initiated the session

send_stop_audio()
Stop an audio streaming session

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

class libpebble2.services.voice.**SetupResult**

FailDisabled = <SetupResult.FailDisabled: 5>

FailTimeout = <SetupResult.FailTimeout: 2>

Success = <SetupResult.Success: 0>

class libpebble2.services.voice.**TranscriptionResult**

FailNoInternet = <TranscriptionResult.FailNoInternet: 1>

FailRecognizerError = <TranscriptionResult.FailRecognizerError: 3>

FailSpeechNotRecognized = <TranscriptionResult.FailSpeechNotRecognized: 4>

Success = <TranscriptionResult.Success: 0>

CHAPTER 5

Exceptions

exception libpebble2.exceptions.**AppInstallError**
Bases: *libpebble2.exceptions.PebbleError*

An app install failed.

exception libpebble2.exceptions.**ConnectionError**
Bases: *libpebble2.exceptions.PebbleError*

Connecting to the Pebble failed.

exception libpebble2.exceptions.**GetBytesError**(*code*)
Bases: *libpebble2.exceptions.PebbleError*

A getbytes session failed.

exception libpebble2.exceptions.**PacketDecodeError**
Bases: *libpebble2.exceptions.PebbleError*

Decoding a packet received from the Pebble failed.

exception libpebble2.exceptions.**PacketEncodeError**
Bases: *libpebble2.exceptions.PebbleError*

Encoding a packet failed.

exception libpebble2.exceptions.**PebbleError**
Bases: exceptions.Exception

The base class for all exceptions raised by libpebble2.

exception libpebble2.exceptions.**PutBytesError**
Bases: *libpebble2.exceptions.PebbleError*

A putbytes session failed.

exception libpebble2.exceptions.**ScreenshotError**
Bases: *libpebble2.exceptions.PebbleError*

A screenshot failed.

exception libpebble2.exceptions.**TimeoutError**
Bases: *libpebble2.exceptions.PebbleError*

Something was waiting for an event and timed out.

CHAPTER 6

Grab bag

This stuff is currently undocumented. All of the following is autogenerated.

6.1 libpebble2.events package

6.1.1 libpebble2.events.mixin module

class libpebble2.events.mixin.**EventSourceMixin**
Bases: object

A convenient mixin to save on repeatedly exposing generic event handler functionality.

register_handler(*event*, *handler*)
Registers a handler to be triggered by an event

Parameters

- **event** – The event to handle
- **handler** – The handler callable.

Returns A handle that can be used to unregister the handler.

unregister_handler(*handle*)
Unregisters an event handler.

Parameters **handle** – The handle returned from *register_handler()*

wait_for_event(*event*, *timeout=10*)
Block waiting for the given event. Returns the event params.

Parameters

- **event** – The event to handle.
- **timeout** – The maximum time to wait before raising *TimeoutError*.

Returns The event params.

6.1.2 libpebble2.events.threaded module

class libpebble2.events.threaded.**ThreadedEventHandler**
Bases: *libpebble2.events.BaseEventHandler*

A threaded implementation of *BaseEventHandler*.

broadcast_event(*event*, **args*)

queue_events(*event*)

register_handler(*event*, *handler*)

unregister_handler(*handle*)

wait_for_event(*event*, *timeout=10*)

6.1.3 Module contents

class libpebble2.events.**BaseEventHandler**
Bases: object

An event handler, used throughout libpebble2 to indicate that something happened. These should ordinarily not need to be directly invoked by a client of libpebble2.

broadcast_event(*event*, **args*)
Broadcasts an event to all subscribers for that event, as added by *register_handler()* *wait_for_event()* and *queue_events()*. All arguments after *event* are passed on to the listeners.

Parameters

- **event** – The event to broadcast.
- **args** – Any arguments to pass on.

queue_events(*event*)
Returns a *BaseEventQueue* from which events can be read as they arrive, even if the arrive faster than they are removed.

Parameters **event** – The events to add to the queue.

Returns An event queue.

Return type *BaseEventQueue*

register_handler(*event*, *handler*)
Register a handler for an event.

Parameters

- **event** – The event to be handled. This can be any object, as long as it's hashable.
- **handler** – A callback function to be called when the event is triggered. The arguments are dependent on the event.

Returns A handle that can be passed to *unregister_handler()* to remove the registration.

unregister_handler(*handle*)
Remove a handler for an event using a handle returned by *register_handler()*.

Parameters **handle** – The handle for the registration to remove.

wait_for_event(*event*, *timeout=10*)
A blocking wait for an event to be fired.

Parameters

- **event** – The event to wait on.
- **timeout** – How long to wait before raising *TimeoutError*

Returns The arguments that were passed to *broadcast_event()*.

class libpebble2.events.**BaseEventQueue**
Bases: object

close()
Stop adding events to this queue. It is illegal to call *get()* or iterate over this queue after calling *close()*.

get(*timeout=10*)
Get the next event in the queue. Blocks until an item is available.

6.2 libpebble2.protocol package

6.2.1 Submodules

6.2.2 libpebble2.protocol.appmessage module

class libpebble2.protocol.appmessage.**AppMessageTuple**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

Represents a tuple in an AppMessage dictionary.

class Type
Bases: enum.IntEnum

AppMessageTuple.**data** = None

AppMessageTuple.**key** = None

AppMessageTuple.**length** = None

AppMessageTuple.**type** = None

class libpebble2.protocol.appmessage.**AppMessagePush**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

count = None

dictionary = None

uuid = None

class libpebble2.protocol.appmessage.**AppMessageACK**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.appmessage.**AppMessageNACK**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.appmessage.**AppMessage**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

transaction_id = None

class libpebble2.protocol.appmessage.**StockAppSetTitle**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class App
Bases: enum.IntEnum

StockAppSetTitle.**app** = None

StockAppSetTitle.**title** = None

class libpebble2.protocol.appmessage.**StockAppSetIcon**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class App
Bases: enum.IntEnum

StockAppSetIcon.**app** = None

StockAppSetIcon.**image_data** = None

StockAppSetIcon.**info_flags** = 4096

StockAppSetIcon.**origin_x** = None

StockAppSetIcon.**origin_y** = None

StockAppSetIcon.**row_size** = None

StockAppSetIcon.**size_x** = None

StockAppSetIcon.**size_y** = None

6.2.3 libpebble2.protocol.apps module

class libpebble2.protocol.apps.**AppRunState**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.apps.**AppRunStateStart**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.apps.**AppRunStateStop**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.apps.**AppRunStateRequest**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.apps.**AppMetadata**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

This represents an entry in the appdb.

app_face_bg_color = None

app_face_template_id = None

app_name = None

app_version_major = None

app_version_minor = None

flags = None

icon = None

sdk_version_major = None

sdk_version_minor = None

uuid = None

6.2.4 libpebble2.protocol.audio module

class libpebble2.protocol.audio.**EncoderFrame**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

data = None

class libpebble2.protocol.audio.**DataTransfer**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

frame_count = None

frames = None

class libpebble2.protocol.audio.**StopTransfer**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.audio.**AudioStream**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

data = None

packet_id = None

session_id = None

6.2.5 libpebble2.protocol.blobdb module

class libpebble2.protocol.blobdb.**InsertCommand**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

key = None

key_size = None

value = None

value_size = None

class libpebble2.protocol.blobdb.**DeleteCommand**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

key = None

key_size = None

class libpebble2.protocol.blobdb.**ClearCommand**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.blobdb.**BlobDatabaseID**
Bases: enum.IntEnum

class libpebble2.protocol.blobdb.**BlobCommand**(**kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

content = None

database = None

token = None

class libpebble2.protocol.blobdb.**BlobStatus**
Bases: enum.IntEnum

class libpebble2.protocol.blobdb.**BlobResponse**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

response = None

token = None

6.2.6 libpebble2.protocol.datalogging module

class libpebble2.protocol.data_logging.**DataLoggingReportOpenSessions**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

sessions = None

class libpebble2.protocol.data_logging.**DataLoggingDespoolOpenSession**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class ItemType
Bases: enum.IntEnum

DataLoggingDespoolOpenSession.**app_uuid** = None

DataLoggingDespoolOpenSession.**data_item_size** = None

DataLoggingDespoolOpenSession.**data_item_type** = None

DataLoggingDespoolOpenSession.**log_tag** = None

DataLoggingDespoolOpenSession.**session_id** = None

DataLoggingDespoolOpenSession.**timestamp** = None

class libpebble2.protocol.data_logging.**DataLoggingDespoolSendData**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

crc = None

data = None

items_left = None

session_id = None

class libpebble2.protocol.data_logging.**DataLoggingCloseSession**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

session_id = None

class libpebble2.protocol.data_logging.**DataLoggingACK**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

session_id = None

class libpebble2.protocol.data_logging.**DataLoggingNACK**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

session_id = None

class libpebble2.protocol.data_logging.**DataLogging**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.data_logging.**DataLoggingTimeout**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.data_logging.**DataLoggingEmptySession**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

session_id = None

class libpebble2.protocol.data_logging.**DataLoggingGetSendEnableRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.data_logging.**DataLoggingGetSendEnableResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

enabled = None

class libpebble2.protocol.data_logging.**DataLoggingSetSendEnable**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

enabled = None

6.2.7 libpebble2.protocol.legacy2 module

class libpebble2.protocol.legacy2.**LegacyNotification**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Source
Bases: enum.IntEnum

LegacyNotification.**body** = None

LegacyNotification.**sender** = None

LegacyNotification.**subject** = None

LegacyNotification.**timestamp** = None

LegacyNotification.**type** = None

class libpebble2.protocol.legacy2.**LegacyBankInfoRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.legacy2.**LegacyRemoveAppUUID**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.legacy2.**LegacyUpgradeAppUUID**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.legacy2.**LegacyAppAvailable**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

bank = None

vibrate = None

class libpebble2.protocol.legacy2.**LegacyListInstalledUUIDs**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.legacy2.**LegacyDescribeInstalledUUID**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.legacy2.**LegacyCurrentAppRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.legacy2.**LegacyAppInstallRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.legacy2.**LegacyBankEntry**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

app_name = None

bank_number = None

company_name = None

flags = None

install_id = None

version_major = None

version_minor = None

class libpebble2.protocol.legacy2.**LegacyBankInfoResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

apps = None

bank_count = None

occupied_banks = None

class libpebble2.protocol.legacy2.**LegacyAppInstallResult**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Status
Bases: enum.IntEnum

LegacyAppInstallResult.**status** = None

class libpebble2.protocol.legacy2.**LegacyAppUUIDsResult**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

count = None

uuids = None

class libpebble2.protocol.legacy2.**LegacyAppDescribeResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

app_name = None

company_name = None

version_major = None

version_minor = None

class libpebble2.protocol.legacy2.**LegacyCurrentAppResponse**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

uuid = None

class libpebble2.protocol.legacy2.**LegacyAppInstallResponse**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.legacy2.**LegacyAppLaunchMessage**(**kwargs)
Bases: *libpebble2.protocol.appmessage.AppMessage*

class Keys
Bases: enum.IntEnum

class LegacyAppLaunchMessage.**States**
Bases: enum.IntEnum

6.2.8 libpebble2.protocol.logs module

class libpebble2.protocol.logs.**RequestLogs**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

generation = None

class libpebble2.protocol.logs.**LogMessage**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

filename = None

length = None

level = None

line = None

message = None

timestamp = None

class libpebble2.protocol.logs.**LogMessageDone**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

class libpebble2.protocol.logs.**NoLogMessages**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

class libpebble2.protocol.logs.**LogDumpShipping**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.logs.**AppLogShippingControl**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

enable = None

class libpebble2.protocol.logs.**AppLogMessage**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

filename = None

level = None

line_number = None

message = None

message_length = None

timestamp = None

uuid = None

6.2.9 libpebble2.protocol.music module

class libpebble2.protocol.music.**MusicControlPlayPause**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlPause**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlPlay**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlNextTrack**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlPreviousTrack**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlVolumeUp**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlVolumeDown**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlGetCurrentTrack**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.music.**MusicControlUpdateCurrentTrack**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

album = None

artist = None

current_track = None

title = None

track_count = None

track_length = None

class libpebble2.protocol.music.**MusicControl**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

6.2.10 libpebble2.protocol.phone module

class libpebble2.protocol.phone.**AnswerCall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**HangUpCall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**PhoneStateRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**IncomingCall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

name = None

number = None

class libpebble2.protocol.phone.**OutgoingCall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**MissedCall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

name = None

number = None

class libpebble2.protocol.phone.**Ring**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**CallStart**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**CallEnd**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.phone.**CallStateItem**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command_id = None

cookie = None

item = None

class libpebble2.protocol.phone.**PhoneStateResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

items = None

class libpebble2.protocol.phone.**PhoneNotification**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command_id = None

cookie = None

message = None

6.2.11 libpebble2.protocol.screenshots module

class libpebble2.protocol.screenshots.**ScreenshotRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.screenshots.**ScreenshotResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

data = None

class libpebble2.protocol.screenshots.**ScreenshotHeader**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class ResponseCode
Bases: enum.IntEnum

ScreenshotHeader.**data** = None

ScreenshotHeader.**height** = None

ScreenshotHeader.**response_code** = None

ScreenshotHeader.**version** = None

ScreenshotHeader.**width** = None

6.2.12 libpebble2.protocol.system module

class libpebble2.protocol.system.**GetTimeRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**GetTimeResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

time = None

class libpebble2.protocol.system.**SetLocaltime**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

time = None

class libpebble2.protocol.system.**SetUTC**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

tz_name = None

unix_time = None

utc_offset = None

class libpebble2.protocol.system.**TimeMessage**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

kind = None

message = None

class libpebble2.protocol.system.**AppVersionRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**AppVersionResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

bugfix_version = None

major_version = None

minor_version = None

platform_flags = None

protocol_caps = None

protocol_version = None

response_version = 2

session_caps = None

class libpebble2.protocol.system.**PhoneAppVersion**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

kind = None

message = None

class libpebble2.protocol.system.**FirmwareUpdateStartResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

response = None

class libpebble2.protocol.system.**SystemMessage**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Type
Bases: enum.IntEnum

SystemMessage.**command** = 0

SystemMessage.**extra_data** = None

SystemMessage.**message_type** = None

class libpebble2.protocol.system.**BLEControl**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

discoverable = None

duration = None

opcode = 4

class libpebble2.protocol.system.**WatchVersionRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**WatchVersionResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

board = None

bootloader_timestamp = None

bt_address = None

capabilities = None

is_unfaithful = None

language = None

language_version = None

recovery = None

resource_crc = None

resource_timestamp = None

running = None

serial = None

class libpebble2.protocol.system.**WatchFirmwareVersion**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

git_hash = None

hardware_platform = None

is_recovery = None

metadata_version = None

timestamp = None

version_tag = None

class libpebble2.protocol.system.**WatchVersion**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.system.**Ping**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

idle = None

class libpebble2.protocol.system.**Pong**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**PingPong**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

cookie = None

message = None

class libpebble2.protocol.system.**Reset**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class **Command**
Bases: enum.IntEnum

Reset.**command** = None

class libpebble2.protocol.system.**Model**
Bases: enum.IntEnum

class libpebble2.protocol.system.**ModelRequest**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**ModelResponse**(**kwargs)
Bases: *libpebble2.protocol.base.PebblePacket*

data = None

length = None

class libpebble2.protocol.system.**ModelError**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class libpebble2.protocol.system.**WatchModel**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

6.2.13 libpebble2.protocol.timeline module

class libpebble2.protocol.timeline.**TimelineAttribute**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

attribute_id = None

content = None

length = None

class libpebble2.protocol.timeline.**TimelineAction**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Type
Bases: enum.IntEnum

TimelineAction.**action_id** = None

TimelineAction.**attribute_count** = None

TimelineAction.**attributes** = None

TimelineAction.**type** = None

class libpebble2.protocol.timeline.**TimelineItem**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Type
Bases: enum.IntEnum

TimelineItem.**action_count** = None

TimelineItem.**actions** = None

TimelineItem.**attribute_count** = None

TimelineItem.**attributes** = None

TimelineItem.**data_length** = None

TimelineItem.**duration** = None

TimelineItem.**flags** = None

TimelineItem.**item_id** = None

TimelineItem.**layout** = None

TimelineItem.**parent_id** = None

TimelineItem.**timestamp** = None

TimelineItem.**type** = None

class libpebble2.protocol.timeline.**TimelineActionEndpoint**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.timeline.**ActionResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Response
Bases: enum.IntEnum

ActionResponse.**attributes** = None

ActionResponse.**item_id** = None

ActionResponse.**num_attributes** = None

ActionResponse.**response** = None

class libpebble2.protocol.timeline.**InvokeAction**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

action_id = None

attributes = None

item_id = None

num_attributes = None

6.2.14 libpebble2.protocol.transfers module

class libpebble2.protocol.transfers.**ObjectType**
Bases: enum.IntEnum

class libpebble2.protocol.transfers.**PutBytesInstall**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

class libpebble2.protocol.transfers.**PutBytesInit**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

bank = None

filename = None

object_size = None

object_type = None

class libpebble2.protocol.transfers.**PutBytesAppInit**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

app_id = None

object_size = None

object_type = None

class libpebble2.protocol.transfers.**PutBytesPut**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

payload = None

payload_size = None

class libpebble2.protocol.transfers.**PutBytesCommit**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

object_crc = None

class libpebble2.protocol.transfers.**PutBytesAbort**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

cookie = None

class libpebble2.protocol.transfers.**PutBytes**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.transfers.**PutBytesApp**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

data = None

class libpebble2.protocol.transfers.**PutBytesResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class Result
Bases: enum.IntEnum

PutBytesResponse.**cookie** = None

PutBytesResponse.**result** = None

class libpebble2.protocol.transfers.**GetBytes**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

command = None

message = None

transaction_id = None

class libpebble2.protocol.transfers.**GetBytesCoredumpRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

Requests a coredump.

class libpebble2.protocol.transfers.**GetBytesDataResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

data = None

offset = None

class libpebble2.protocol.transfers.**GetBytesFileRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

Requests a file. This only works on non-release firmwares.

filename = None

class libpebble2.protocol.transfers.**GetBytesInfoResponse**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

class **ErrorCode**
Bases: enum.IntEnum

GetBytesInfoResponse.**error_code** = None

GetBytesInfoResponse.**num_bytes** = None

class libpebble2.protocol.transfers.**GetBytesFlashRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

Requests a region of flash. This only works on non-release firmwares.

length = None

offset = None

class libpebble2.protocol.transfers.**GetBytesUnreadCoredumpRequest**(***kwargs*)
Bases: *libpebble2.protocol.base.PebblePacket*

Requests a coredump, but errors if it has already been read.

6.3 libpebble2.util package

6.3.1 libpebble2.util.bundle module

class libpebble2.util.bundle.**PebbleBundle**(*bundle_path*, *hardware=0*)
Bases: object

MANIFEST_FILENAME = 'manifest.json'

PLATFORM_PATHS = {'aplite': ('aplite/', ''), 'unknown': ('',), 'emery': ('emery/', 'basalt/', ''), 'basalt': ('basalt/', ''), 'cha

STRUCT_DEFINITION = ['8s', '2B', '2B', '2B', 'H', 'I', 'I', '32s', '32s', 'I', 'I', 'I', 'I', '16s']

UNIVERSAL_FILES = set(['pebble-js-app.js', 'appinfo.json'])

close()

get_app_metadata()

get_app_path()

get_application_info()

get_firmware_info()

get_manifest()

get_real_path(*path*)

get_resource_path()

get_resources_info()

get_worker_info()

get_worker_path()

has_javascript

has_resources

has_worker

is_app_bundle

is_firmware_bundle

classmethod prefixes_for_hardware(*hardware*)

6.3.2 libpebble2.util.hardware module

class libpebble2.util.hardware.**PebbleHardware**

Bases: object

BIANCA = 6

BOBBY_SMILES = 10

PLATFORMS = {0: 'unknown', 1: 'aplite', 2: 'aplite', 3: 'aplite', 4: 'aplite', 5: 'aplite', 6: 'aplite', 7: 'basalt', 8: 'basalt', 9

ROBERT_BB = 249

ROBERT_EVT = 13

SILK = 14

SILK_BB = 250

SILK_BB2 = 248

SILK_EVT = 12

SNOWY_BB = 253

SNOWY_BB2 = 252

SNOWY_DVT = 8

SNOWY_EVT2 = 7

SPALDING = 11

SPALDING_BB2 = 251

SPALDING_EVT = 9

TINTIN_BB = 255

TINTIN_BB2 = 254

TINTIN_EV1 = 1

TINTIN_EV2 = 2

TINTIN_EV2_3 = 3

TINTIN_EV2_4 = 4

TINTIN_V1_5 = 5

UNKNOWN = 0

classmethod hardware_platform(*hardware*)

6.3.3 libpebble2.util.stm32_crc module

libpebble2.util.stm32_crc.**crc32**(*data*)

libpebble2.util.stm32_crc.**process_buffer**(*buf*, *c=4294967295*)

libpebble2.util.stm32_crc.**process_word**(*data*, *crc=4294967295*)

libpebble2 is a python library for interacting with Pebble devices. It:

- Supports connections to Pebble QEMU instances and to watches via the Pebble mobile app
- Supports connection to watches running both 2.x and 3.x firmware on aplite or basalt hardware
- Provides automatic serialisation and deserialisation of pebble protocol messages
- Asynchronous information is provided by a usable event system
- Features a simple DSL for defining new message types
- Provides ready-made implementations several Pebble Protocol services, including BlobDB and app installation
- Works on Python 2.7 and 3.4

CHAPTER 7

Getting Started

7.1 Installation

`pip install libpebble2`, or grab the source from https://github.com/pebble/libpebble2

7.2 Usage

Connecting:

```
>>> from libpebble2.communication import PebbleConnection
>>> from libpebble2.communication.transports.websocket import WebsocketTransport
>>> pebble = PebbleConnection(WebsocketTransport("ws://192.168.0.204:9000/"))
>>> pebble.connect()
>>> pebble.run_async()
>>> pebble.watch_info.serial
u'Q306175E006V'
```

Sending and receiving messages:

```
>>> from libpebble2.protocol import *
>>> pebble.send_packet(PingPong(message=Ping(), cookie=53))
>>> pebble.read_from_endpoint(PingPong)
PingPong(command=1, cookie=53, message=Pong())
```

Installing an app:

```
>>> from libpebble2.services.install import AppInstaller
>>> AppInstaller(pebble, "some_app.pbw").install()
```

CHAPTER 8

Components

libpebble2 is split into a number of components.

8.1 Communication and transports

libpebble2 provides a *PebbleConnection* to connect to a Pebble. This class manages all Pebble Protocol communication, but does not itself know how to establish a connection to one. Connecting to a Pebble is handled by the transports, *QemuTransport* and *WebsocketTransport*. It is possible to define new transports if necessary.

8.2 Protocol

The protocol layer provides serialisation and deserialisation of Pebble Protocol messages (and, in fact, any arbitrary packed structure). It provides a simple DSL for defining messages:

```
class WatchFirmwareVersion(PebblePacket):
    timestamp = Uint32()
    version_tag = FixedString(32)
    git_hash = FixedString(8)
    is_recovery = Boolean()
    hardware_platform = Uint8()
    metadata_version = Uint8()
```

Most messages are defined by the library in the `protocol` package, but defining more is easy.

8.3 Services

Some watch services are more complex than one or two messages. For these, services are provided to reduce effort.

```
>>> import png  # from pypng
>>> image = Screenshot(pebble).grab_image()
>>> png.from_array(image, mode="RGB;8").save("screenshot.png")
```

CHAPTER 9

Indices and tables

- genindex
- modindex

l

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W