

---

# **tellcore-py Documentation**

***Release 1.1.3***

**Erik Johansson**

**Jun 27, 2023**



---

## Contents

---

<b>1</b>	<b>telldus.telldus (module)</b>	<b>3</b>
1.1	TelldusCore . . . . .	3
1.2	DeviceFactory . . . . .	5
1.3	Device . . . . .	5
1.4	DeviceGroup . . . . .	6
1.5	Sensor . . . . .	6
1.6	SensorValue . . . . .	7
1.7	Controller . . . . .	7
1.8	QueuedCallbackDispatcher . . . . .	7
1.9	AsyncioCallbackDispatcher . . . . .	8
<b>2</b>	<b>telldus.constants (module)</b>	<b>9</b>
<b>3</b>	<b>telldus.library (module)</b>	<b>11</b>
3.1	Library . . . . .	11
3.2	TelldusError . . . . .	12
3.3	BaseCallbackDispatcher . . . . .	12
3.4	DirectCallbackDispatcher . . . . .	13
<b>4</b>	<b>telldus_tool (example)</b>	<b>15</b>
<b>5</b>	<b>Event handling (example)</b>	<b>19</b>
<b>6</b>	<b>Changelog</b>	<b>23</b>
6.1	1.1.3 (2016-11-22) . . . . .	23
6.2	1.1.2 (2015-06-24) . . . . .	23
6.3	1.1.1 (2015-05-01) . . . . .	23
6.4	1.1.0 (2014-11-19) . . . . .	23
6.5	1.0.4 (2014-11-05) . . . . .	23
6.6	1.0.3 (2014-02-02) . . . . .	24
6.7	1.0.2 (2014-01-28) . . . . .	24
6.8	1.0.1 (2014-01-26) . . . . .	24
6.9	1.0.0 (2014-01-09) . . . . .	24
6.10	0.9.0 (2014-01-03) . . . . .	24
6.11	0.8.0 (2013-08-11) . . . . .	24
6.12	0.1.0 (2013-06-26) . . . . .	24

<b>7 Indices and tables</b>	<b>25</b>
<b>Python Module Index</b>	<b>27</b>
<b>Index</b>	<b>29</b>

telldus-core is a Python wrapper for [Telldus](#)' home automation library [Telldus Core](#).

Contents:



---

## tellcore.telldus (module)

---

This module provides a high level Python interface to Telldus' C API.

Since most functions are Python-ified wrappers around the C API, please also refer to the [Telldus Core documentation](#) for further information.

The classes in this module all use the `library.Library` class under the hood, and thus also has e.g. automatic memory management, exceptions (`library.TelldusError`) and transparent string conversion with full Python 3 support.

Some example programs are included in the documentation to help understand how to use the different classes:

- [telldus\\_tool \(example\)](#)
- [Event handling \(example\)](#)

## 1.1 TelldusCore

**class** `tellcore.telldus.TelldusCore` (*library\_path=None, callback\_dispatcher=None*)

The main class for tellcore-py.

Has methods for adding devices and for enumerating controllers, devices and sensors. Also handles callbacks; both registration and making sure the callbacks are processed in the main thread instead of the callback thread.

**\_\_init\_\_** (*library\_path=None, callback\_dispatcher=None*)

Create a new TelldusCore instance.

Only one instance should be used per program.

### Parameters

- **library\_path** (*str*) – Passed to the `library.Library` constructor.
- **callback\_dispatcher** (*str*) – An instance implementing the `library.BaseCallbackDispatcher` interface ( e.g. `QueuedCallbackDispatcher` or `AsyncioCallbackDispatcher`) A callback dispatcher must be provided if callbacks are to be used.

**callback\_dispatcher**

The callback dispatcher used. Set when constructing the instance and should not be changed.

**add\_device** (*name, protocol, model=None, \*\*parameters*)

Add a new device.

**Returns** a *Device* or *DeviceGroup* instance.

**add\_group** (*name, devices*)

Add a new device group.

**Returns** a *DeviceGroup* instance.

**connect\_controller** (*vid, pid, serial*)

Connect a controller.

**controllers** ()

Return all known controllers.

Requires Telldus core library version >= 2.1.2.

**Returns** list of *Controller* instances.

**devices** ()

Return all known devices.

**Returns** list of *Device* or *DeviceGroup* instances.

**disconnect\_controller** (*vid, pid, serial*)

Disconnect a controller.

**register\_controller\_event** (*callback*)

Register a new controller event callback handler.

See *Event handling (example)* for more information.

**Returns** the callback id

**register\_device\_change\_event** (*callback*)

Register a new device change event callback handler.

See *Event handling (example)* for more information.

**Returns** the callback id

**register\_device\_event** (*callback*)

Register a new device event callback handler.

See *Event handling (example)* for more information.

**Returns** the callback id

**register\_raw\_device\_event** (*callback*)

Register a new raw device event callback handler.

See *Event handling (example)* for more information.

**Returns** the callback id

**register\_sensor\_event** (*callback*)

Register a new sensor event callback handler.

See *Event handling (example)* for more information.

**Returns** the callback id



**send\_raw\_command** (*command*, *reserved*=0)

Send a raw command.

**sensors** ()

Return all known sensors.

**Returns** list of *Sensor* instances.

**unregister\_callback** (*cid*)

Unregister a callback handler.

**Parameters** *id* (*int*) – the callback id as returned from one of the *register\_\*\_event* methods.

## 1.2 DeviceFactory

`telcore.telldus.DeviceFactory` (*id*, *lib*=None)

Create the correct device instance based on device type and return it.

**Returns** a *Device* or *DeviceGroup* instance.

## 1.3 Device

**class** `telcore.telldus.Device` (*id*, *lib*=None)

A device that can be controlled by Telldus Core.

Can be instantiated directly if the id is known, but using *DeviceFactory*() is recommended. Otherwise returned from *TelldusCore.add\_device()* or *TelldusCore.devices()*.

**name**

The name of the device (read/write).

**protocol**

The protocol used for the device (read/write).

**model**

The device's model (read/write).

**type**

The device type (read only). One of the device type constants from *telcore.constants*.

**bell** ()

Send “bell” command to the device.

**dim** (*level*)

Dim the device.

**Parameters** *level* (*int*) – The level to dim to in the range [0, 255].

**down** ()

Send “down” command to the device.

**execute** ()

Send “execute” command to the device.

**get\_parameter** (*name*)

Get a parameter.

**last\_sent\_command** (*methods\_supported*)

Get the last sent (or seen) command.

**last\_sent\_value** ()  
Get the last sent (or seen) value.

**learn** ()  
Send “learn” command to the device.

**methods** (*methods\_supported*)  
Query the device for supported methods.

**parameters** ()  
Get dict with all set parameters.

**remove** ()  
Remove the device from Telldus Core.

**set\_parameter** (*name, value*)  
Set a parameter.

**stop** ()  
Send “stop” command to the device.

**turn\_off** ()  
Turn off the device.

**turn\_on** ()  
Turn on the device.

**up** ()  
Send “up” command to the device.

## 1.4 DeviceGroup

**class** tellcore.telldus.**DeviceGroup** (*id, lib=None*)  
Extends *Device* with methods for managing a group

E.g. when a group is turned on, all devices in that group are turned on.

**add\_to\_group** (*devices*)  
Add device(s) to the group.

**devices\_in\_group** ()  
Fetch list of devices in group.

**remove\_from\_group** (*devices*)  
Remove device(s) from the group.

## 1.5 Sensor

**class** tellcore.telldus.**Sensor** (*protocol, model, id, datatypes, lib=None*)  
Represents a sensor.

Returned from *TelldusCore.sensors* ()

**has\_value** (*datatype*)  
Return True if the sensor supports the given data type.

sensor.has\_value(TELLSTICK\_TEMPERATURE) is identical to calling sensor.has\_temperature().

**value** (*datatype*)

Return the *SensorValue* for the given data type.

`sensor.value(TELLSTICK_TEMPERATURE)` is identical to calling `sensor.temperature()`.

## 1.6 SensorValue

**class** `telcore.telldus.SensorValue` (*datatype, value, timestamp*)

Represents a single sensor value.

Returned from `Sensor.value()`.

**datatype**

One of the sensor value type constants from `telcore.constants`.

**value**

The sensor value.

**timestamp**

The time the sensor value was registered (in seconds since epoch).

## 1.7 Controller

**class** `telcore.telldus.Controller` (*id, type, lib=None*)

Represents a Telldus controller.

Returned from `TelldusCore.controllers()`

**id**

**type**

One of the controller type constants from `telcore.constants`.

## 1.8 QueuedCallbackDispatcher

**class** `telcore.telldus.QueuedCallbackDispatcher`

The default callback dispatcher used by `TelldusCore`.

Queues callbacks that arrive from Telldus Core. Then calls them in the main thread (or more precise: the thread calling `process_callback()`) instead of the callback thread used by Telldus Core. This way the application using `TelldusCore` don't have to do any thread synchronization. Only make sure `process_pending_callbacks()` is called regularly.

**on\_callback** (*callback, \*args*)

Called from the callback thread when an event is received.

**Parameters**

- **callback** (*callable*) – The callback function to call.
- **args** – The arguments to pass to the callback.

**process\_callback** (*block=True*)

Dispatch a single callback in the current thread.

**Parameters** **block** (*boolean*) – If True, blocks waiting for a callback to come.

**Returns** True if a callback was processed; otherwise False.

**process\_pending\_callbacks** ()

Dispatch all pending callbacks in the current thread.

## 1.9 AsyncioCallbackDispatcher

**class** tellcore.telldus.**AsyncioCallbackDispatcher** (*loop*)

Dispatcher for use with the event loop available in Python 3.4+.

Callbacks will be dispatched on the thread running the event loop. The loop argument should be a BaseEvent-Loop instance, e.g. the one returned from `asyncio.get_event_loop()`.

**on\_callback** (*callback*, \**args*)

Called from the callback thread when an event is received.

### Parameters

- **callback** (*callable*) – The callback function to call.
- **args** – The arguments to pass to the callback.

## CHAPTER 2

---

### tellcore.constants (module)

---

Contains all constants used in the API.

All the TELLSTICK\_\* defines documented in the [Telldus Core documentation](#) are available here.

**The recommended usage is to do::** import tellcore.constants as const

The constants will then be available as const.TELLSTICK\_TURNON, etc. See below for the full list of all available constants.

```
# Device methods
TELLSTICK_TURNON = 1
TELLSTICK_TURNOFF = 2
TELLSTICK_BELL = 4
TELLSTICK_TOGGLE = 8
TELLSTICK_DIM = 16
TELLSTICK_LEARN = 32
TELLSTICK_EXECUTE = 64
TELLSTICK_UP = 128
TELLSTICK_DOWN = 256
TELLSTICK_STOP = 512

# Sensor value types
TELLSTICK_TEMPERATURE = 1
TELLSTICK_HUMIDITY = 2
TELLSTICK_RAINRATE = 4
TELLSTICK_RAINTOTAL = 8
TELLSTICK_WINDDIRECTION = 16
TELLSTICK_WINDAVERAGE = 32
TELLSTICK_WINDGUST = 64

# Error codes
TELLSTICK_SUCCESS = 0
TELLSTICK_ERROR_NOT_FOUND = -1
TELLSTICK_ERROR_PERMISSION_DENIED = -2
TELLSTICK_ERROR_DEVICE_NOT_FOUND = -3
```

(continues on next page)

(continued from previous page)

```
TELLSTICK_ERROR_METHOD_NOT_SUPPORTED = -4
TELLSTICK_ERROR_COMMUNICATION = -5
TELLSTICK_ERROR_CONNECTING_SERVICE = -6
TELLSTICK_ERROR_UNKNOWN_RESPONSE = -7
TELLSTICK_ERROR_SYNTAX = -8
TELLSTICK_ERROR_BROKEN_PIPE = -9
TELLSTICK_ERROR_COMMUNICATING_SERVICE = -10
TELLSTICK_ERROR_CONFIG_SYNTAX = -11
TELLSTICK_ERROR_UNKNOWN = -99

# Device types
TELLSTICK_TYPE_DEVICE = 1
TELLSTICK_TYPE_GROUP = 2
TELLSTICK_TYPE_SCENE = 3

# Controller types
TELLSTICK_CONTROLLER_TELLSTICK = 1
TELLSTICK_CONTROLLER_TELLSTICK_DUO = 2
TELLSTICK_CONTROLLER_TELLSTICK_NET = 3

# Device changes
TELLSTICK_DEVICE_ADDED = 1
TELLSTICK_DEVICE_CHANGED = 2
TELLSTICK_DEVICE_REMOVED = 3
TELLSTICK_DEVICE_STATE_CHANGED = 4

# Change types
TELLSTICK_CHANGE_NAME = 1
TELLSTICK_CHANGE_PROTOCOL = 2
TELLSTICK_CHANGE_MODEL = 3
TELLSTICK_CHANGE_METHOD = 4
TELLSTICK_CHANGE_AVAILABLE = 5
TELLSTICK_CHANGE_FIRMWARE = 6
```

---

## tellcore.library (module)

---

The classes in this module are not meant to be used directly. They are mostly support classes for the higher level API described in [tellcore.telldus \(module\)](#).

### 3.1 Library

**class** tellcore.library.**Library** (*name=None, callback\_dispatcher=None*)

Wrapper around the Telldus Core C API.

With the exception of `tdInit`, `tdClose` and `tdReleaseString`, all functions in the C API (see [Telldus Core documentation](#)) can be called. The parameters are the same as in the C API documentation. The return value are mostly the same as for the C API, except for functions with multiple out parameters.

**In addition, this class:**

- automatically frees memory for strings returned from the C API,
- converts errors returned from functions into ([TelldusError](#)) exceptions,
- transparently converts between Python strings and C style strings.

**\_\_init\_\_** (*name=None, callback\_dispatcher=None*)

Load and initialize the Telldus core library.

The underlying library is only initialized the first time this object is created. Subsequent instances uses the same underlying library instance.

#### Parameters

- **name** (*str*) – If None than the platform specific name of the Telldus library is used, but it can be e.g. an absolute path.
- **callback\_dispatcher** – If callbacks are to be used, this parameter must refer to an instance of a class inheriting from [BaseCallbackDispatcher](#).

**\_\_del\_\_** ()

Close and unload the Telldus core library.

Any callback set up is removed.

The underlying library is only closed and unloaded if this is the last instance sharing the same underlying library instance.

**class** `c_string_p` (*param*)

**classmethod** `from_param` (*param*)

Convert a Python object into a function call parameter.

**tdController** ()

Get the next controller while iterating.

**Returns** a dict with the keys: id, type, name, available.

**tdSensor** ()

Get the next sensor while iterating.

**Returns** a dict with the keys: protocol, model, id, datatypes.

**tdSensorValue** (*protocol, model, sid, datatype*)

Get the sensor value for a given sensor.

**Returns** a dict with the keys: value, timestamp.

## 3.2 TelldusError

**exception** `tellcore.library.TelldusError` (*error, lib=None*)

Error returned from Telldus Core API.

Automatically raised when a function in the C API returns an error.

**Attributes:** `error`: The error code constant (one of `TELLSTICK_ERROR_*` from `tellcore.constants`).

**\_\_str\_\_** ()

Return the human readable error string.

## 3.3 BaseCallbackDispatcher

**class** `tellcore.library.BaseCallbackDispatcher`

Base callback dispatcher class.

Inherit from this class and override the `on_callback()` method to change how callbacks are dispatched.

**on\_callback** (*callback, \*args*)

Called from the callback thread when an event is received.

**Parameters**

- **callback** (*callable*) – The callback function to call.
- **args** – The arguments to pass to the callback.



## 3.4 DirectCallbackDispatcher

**class** `tellcore.library.DirectCallbackDispatcher`

Dispatches callbacks directly.

Since the callback is dispatched directly, the callback is called in the callback thread.

**on\_callback** (*callback*, \**args*)

Called from the callback thread when an event is received.

### Parameters

- **callback** (*callable*) – The callback function to call.
- **args** – The arguments to pass to the callback.



## CHAPTER 4

---

### tellcore\_tool (example)

---

The example below is an implementation of `tdtool` similar to the one included in the official Tellus Core distribution.

```
#!/usr/bin/env python
# Copyright (c) 2014 Erik Johansson <erik@ejohansson.se>
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 3 of the
# License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA

import argparse
import sys
import time

import tellcore.telldus as td
import tellcore.constants as const

if sys.version_info < (3, 0):
    import tellcore.library as lib
    lib.Library.DECODE_STRINGS = False

def print_devices(devices):
    print("Number of devices: {}\n".format(len(devices)))
    print("{:<5s} {:<25s} {:<10s} {:<10s} {:<20s} {}".format(
```

(continues on next page)

(continued from previous page)

```

        "ID", "NAME", "STATE", "PROTOCOL", "MODEL", "PARAMETERS"))
for device in devices:
    cmd = device.last_sent_command(
        const.TELLSTICK_TURNON
        | const.TELLSTICK_TURNOFF
        | const.TELLSTICK_DIM)
    if cmd == const.TELLSTICK_TURNON:
        cmd_str = "ON"
    elif cmd == const.TELLSTICK_TURNOFF:
        cmd_str = "OFF"
    elif cmd == const.TELLSTICK_DIM:
        cmd_str = "DIMMED:{}".format(device.last_sent_value())
    else:
        cmd_str = "UNKNOWN:{}".format(cmd)
    params_str = ""
    for name, value in device.parameters().items():
        params_str += " {}:{}".format(name, value)
    print("{:<5d} {:<25s} {:<10s} {:<10s} {:<20s}{}".format(
        device.id, device.name, cmd_str,
        device.protocol, device.model, params_str))

def print_sensors(sensors):
    print("Number of sensors: {}".format(len(sensors)))
    print("{:<15s} {:<15s} {:<5s} {:<8s} {:<8s} {:<18s} {:<20s} {}".format(
        "PROTOCOL", "MODEL", "ID", "TEMP", "HUMIDITY", "RAIN", "WIND",
        "LAST UPDATED"))

def format_value(sensor, datatype, formatter):
    if not sensor.has_value(datatype):
        return ("", None)
    value = sensor.value(datatype)
    return (formatter(value.value), value.datetime)

for sensor in sensors:
    values = []
    values.append(format_value(sensor, const.TELLSTICK_TEMPERATURE,
                               lambda x: "{} C".format(x)))
    values.append(format_value(sensor, const.TELLSTICK_HUMIDITY,
                               lambda x: "{} %".format(x)))
    values.append(format_value(sensor, const.TELLSTICK_RAINRATE,
                               lambda x: x + " mm/h "))
    values.append(format_value(sensor, const.TELLSTICK_RAINTOTAL,
                               lambda x: x + " mm"))
    values.append(format_value(sensor, const.TELLSTICK_WINDDIRECTION,
                               lambda x: ["N", "NNE", "NE", "ENE",
                                           "E", "ESE", "SE", "SSE",
                                           "S", "SSW", "SW", "WSW",
                                           "W", "WNW", "NW", "NNW"]
                                           [int(float(x) / 22.5)] + " "))
    values.append(format_value(sensor, const.TELLSTICK_WINDAVERAGE,
                               lambda x: x + " m/s "))
    values.append(format_value(sensor, const.TELLSTICK_WINDGUST,
                               lambda x: "{} m/s".format(x)))

    # Get first valid timestamp
    timestamp = [v[1] for v in values if v[1] is not None][0]

```

(continues on next page)

(continued from previous page)

```

        s = [v[0] for v in values]
        values_str = "{:<8s} {:<8s} ".format(s[0], s[1])
        values_str += "{:<18s} ".format(s[2] + s[3])
        values_str += "{:<20s} ".format(s[4] + s[5] + s[6])

        print("{:<15s} {:<15s} {:<5d} {}".format(
            sensor.protocol, sensor.model, sensor.id, values_str,
            timestamp))

def find_device(device, devices):
    for d in devices:
        if str(d.id) == device or d.name == device:
            return d
    print("Device '{}' not found".format(device))
    return None

parser = argparse.ArgumentParser(
    description='Telldus administration tool',
    epilog='DEVICE can be either device id or name')
group = parser.add_mutually_exclusive_group(required=True)

group.add_argument(
    '-l', '--list', action='store_true',
    help='List all configured devices and discovered sensors')
group.add_argument(
    '--list-devices', action='store_true',
    help='List all configured devices')
group.add_argument(
    '--list-sensors', action='store_true',
    help='List all discovered sensors')
group.add_argument(
    '--on', metavar='DEVICE', help='Turn on device')
group.add_argument(
    '--off', metavar='DEVICE', help='Turn off device')
group.add_argument(
    '--learn', metavar='DEVICE', help='Send learn command to device')
group.add_argument(
    '--remove', metavar='DEVICE', help='Remove device')

args = vars(parser.parse_args())

core = td.TelldusCore()

if args['list']:
    print_devices(core.devices())
    print("")
    print_sensors(core.sensors())
elif args['list_devices']:
    print_devices(core.devices())
elif args['list_sensors']:
    print_sensors(core.sensors())
elif args['on'] is not None:
    device = find_device(args['on'], core.devices())
    if device is not None:
        device.turn_on()
elif args['off'] is not None:
    device = find_device(args['off'], core.devices())

```

(continues on next page)

(continued from previous page)

```
    if device is not None:
        device.turn_off()
elif args['learn'] is not None:
    device = find_device(args['learn'], core.devices())
    if device is not None:
        device.learn()
elif args['remove'] is not None:
    device = find_device(args['remove'], core.devices())
    if device is not None:
        device.remove()
```

---

## Event handling (example)

---

The example below illustrates how event callbacks are registered and processed using the default `tellusdus.QueuedCallbackDispatcher` dispatcher.

For more information regarding the arguments to the callback functions, please refer to the official [Telldus Core documentation](#) (see the callback typedefs). Please note that the context mentioned there is not included in `tellcore-py`.

```
#!/usr/bin/env python
# Copyright (c) 2012-2014 Erik Johansson <erik@ejohansson.se>
#
# This program is free software; you can redistribute it and/or
# modify it under the terms of the GNU General Public License as
# published by the Free Software Foundation; either version 3 of the
# License, or (at your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA

import argparse
import sys

import tellcore.telldus as td
import tellcore.constants as const

METHODS = {const.TELLSTICK_TURNON: 'turn on',
            const.TELLSTICK_TURNOFF: 'turn off',
            const.TELLSTICK_BELL: 'bell',
            const.TELLSTICK_TOGGLE: 'toggle',
```

(continues on next page)

(continued from previous page)

```

        const.TELLSTICK_DIM: 'dim',
        const.TELLSTICK_LEARN: 'learn',
        const.TELLSTICK_EXECUTE: 'execute',
        const.TELLSTICK_UP: 'up',
        const.TELLSTICK_DOWN: 'down',
        const.TELLSTICK_STOP: 'stop'}

EVENTS = {const.TELLSTICK_DEVICE_ADDED: "added",
          const.TELLSTICK_DEVICE_REMOVED: "removed",
          const.TELLSTICK_DEVICE_CHANGED: "changed",
          const.TELLSTICK_DEVICE_STATE_CHANGED: "state changed"}

CHANGES = {const.TELLSTICK_CHANGE_NAME: "name",
            const.TELLSTICK_CHANGE_PROTOCOL: "protocol",
            const.TELLSTICK_CHANGE_MODEL: "model",
            const.TELLSTICK_CHANGE_METHOD: "method",
            const.TELLSTICK_CHANGE_AVAILABLE: "available",
            const.TELLSTICK_CHANGE_FIRMWARE: "firmware"}

TYPES = {const.TELLSTICK_CONTROLLER_TELLSTICK: 'tellstick',
         const.TELLSTICK_CONTROLLER_TELLSTICK_DUO: "tellstick duo",
         const.TELLSTICK_CONTROLLER_TELLSTICK_NET: "tellstick net"}

def device_event(id_, method, data, cid):
    method_string = METHODS.get(method, "UNKNOWN METHOD {0}".format(method))
    string = "[DEVICE] {0} -> {1}".format(id_, method_string)
    if method == const.TELLSTICK_DIM:
        string += " [{0}]".format(data)
    print(string)

def device_change_event(id_, event, type_, cid):
    event_string = EVENTS.get(event, "UNKNOWN EVENT {0}".format(event))
    string = "[DEVICE_CHANGE] {0} {1}".format(event_string, id_)
    if event == const.TELLSTICK_DEVICE_CHANGED:
        type_string = CHANGES.get(type_, "UNKNOWN CHANGE {0}".format(type_))
        string += " [{0}]".format(type_string)
    print(string)

def raw_event(data, controller_id, cid):
    string = "[RAW] {0} <- {1}".format(controller_id, data)
    print(string)

def sensor_event(protocol, model, id_, dataType, value, timestamp, cid):
    string = "[SENSOR] {0} [{1}/{2}] ({3}) @ {4} <- {5}".format(
        id_, protocol, model, dataType, timestamp, value)
    print(string)

def controller_event(id_, event, type_, new_value, cid):
    event_string = EVENTS.get(event, "UNKNOWN EVENT {0}".format(event))
    string = "[CONTROLLER] {0} {1}".format(event_string, id_)
    if event == const.TELLSTICK_DEVICE_ADDED:
        type_string = TYPES.get(type_, "UNKNOWN TYPE {0}".format(type_))

```

(continues on next page)



(continued from previous page)

```

        string += " {0}".format(type_string)
    elif (event == const.TELLSTICK_DEVICE_CHANGED
          or event == const.TELLSTICK_DEVICE_STATE_CHANGED):
        type_string = CHANGES.get(type_, "UNKNOWN CHANGE {0}".format(type_))
        string += " [{0}] -> {1}".format(type_string, new_value)
    print(string)

parser = argparse.ArgumentParser(description='Listen for Telldus events.')

parser.add_argument(
    '--all', action='store_true', help='Trace all events')
parser.add_argument(
    '--device', action='store_true', help='Trace device events')
parser.add_argument(
    '--change', action='store_true', help='Trace device change events')
parser.add_argument(
    '--raw', action='store_true', help='Trace raw events')
parser.add_argument(
    '--sensor', action='store_true', help='Trace sensor events')
parser.add_argument(
    '--controller', action='store_true', help='Trace controller events')

args = vars(parser.parse_args())

try:
    import asyncio
    loop = asyncio.get_event_loop()
    dispatcher = td.AsyncioCallbackDispatcher(loop)
except ImportError:
    loop = None
    dispatcher = td.QueuedCallbackDispatcher()

core = td.TelldusCore(callback_dispatcher=dispatcher)
callbacks = []

for arg in args:
    if not (args[arg] or args['all']):
        continue
    try:
        if arg == 'device':
            callbacks.append(core.register_device_event(device_event))
        elif arg == 'change':
            callbacks.append(
                core.register_device_change_event(device_change_event))
        elif arg == 'raw':
            callbacks.append(core.register_raw_device_event(raw_event))
        elif arg == 'sensor':
            callbacks.append(core.register_sensor_event(sensor_event))
        elif arg == 'controller':
            callbacks.append(core.register_controller_event(controller_event))
        else:
            assert arg == 'all'
    except AttributeError:
        if not args['all']:
            raise

```

(continues on next page)

(continued from previous page)

```
if len(callbacks) == 0:
    print("Must enable at least one event")
    parser.print_usage()
    sys.exit(1)

try:
    if loop:
        loop.run_forever()
    else:
        import time
        while True:
            core.callback_dispatcher.process_pending_callbacks()
            time.sleep(0.5)
except KeyboardInterrupt:
    pass
```

### 6.1 1.1.3 (2016-11-22)

- Added datetime attribute to `SensorValue`.
- Fixed strange problem in `Library` where the class itself could sometimes be `None` in `__del__`.

### 6.2 1.1.2 (2015-06-24)

- Added option to `Library` to make it possible to select if strings should be decoded or not.
- Made `tellcore_tool` not decode strings (i.e. convert to unicode) when running under Python 2.x to avoid unicode errors when printing non ascii characters.

### 6.3 1.1.1 (2015-05-01)

- Fixed a bug that made `tellcore_tool` not work with Python 3.x.

### 6.4 1.1.0 (2014-11-19)

- The callback dispatcher is no longer global, but tied to a `Library` instance. Applications wishing to use callbacks must now pass an explicit dispatcher instance to the `TelldusCore` constructor.

### 6.5 1.0.4 (2014-11-05)

- Made `last_sent_value` return an int instead of string.

## 6.6 1.0.3 (2014-02-02)

- Work around crash in Telldus Core (< v2.1.2) when re-initializing the library after `tdClose`.

## 6.7 1.0.2 (2014-01-28)

- Packaging fixes.

## 6.8 1.0.1 (2014-01-26)

- Added `AsyncioCallbackDispatcher` class for integrating callbacks with the new event loop available in Python 3.4 (asyncio).
- Include tools from `bin/` when installing.

## 6.9 1.0.0 (2014-01-09)

- Added high level support for device groups in the form of the new class `DeviceGroup`.
- More complete documentation.
- Removed the methods `process_callback` and `process_pending_callbacks` from `TelldusCore`. Instead, `callback_dispatcher` is now a public attribute of `TelldusCore` and the default callback dispatcher `QueuedCallbackDispatcher` implements the two methods instead.

## 6.10 0.9.0 (2014-01-03)

- Telldus functions that used to return `bool` (`tdSetName`, `tdSetProtocol`, `tdSetModel`, `tdSetDeviceParameter` and `tdRemoveDevice`) now raise an exception instead of returning `False`.
- Support for rain- and windsensors.
- Include data type in `SensorValue`.

## 6.11 0.8.0 (2013-08-11)

- Improved callback handling to simplify integration with different event loops. Parameter conversion is now done in the library code and the adaptation to different event loops is done by a simple callback dispatch class. The default dispatcher (when using `TelldusCore`) is still done using a queue.
- New documentation for parts of the package. Can be read online at <https://tellcore-py.readthedocs.org/>.
- Fix problem with strings and python 3 (issue #2).

## 6.12 0.1.0 (2013-06-26)

- First release.

## CHAPTER 7

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### t

`tellcore.constants`, 9  
`tellcore.library`, 11  
`tellcore.telldus`, 3





## Symbols

`__del__()` (*tellcore.library.Library* method), 11  
`__init__()` (*tellcore.library.Library* method), 11  
`__init__()` (*tellcore.telldus.TelldusCore* method), 3  
`__str__()` (*tellcore.library.TelldusError* method), 12

## A

`add_device()` (*tellcore.telldus.TelldusCore* method), 4  
`add_group()` (*tellcore.telldus.TelldusCore* method), 4  
`add_to_group()` (*tellcore.telldus.DeviceGroup* method), 6  
`AsyncioCallbackDispatcher` (class in *tellcore.telldus*), 8

## B

`BaseCallbackDispatcher` (class in *tellcore.library*), 12  
`bell()` (*tellcore.telldus.Device* method), 5

## C

`callback_dispatcher` (*tellcore.telldus.TelldusCore* attribute), 3  
`connect_controller()` (*tellcore.telldus.TelldusCore* method), 4  
`Controller` (class in *tellcore.telldus*), 7  
`controllers()` (*tellcore.telldus.TelldusCore* method), 4

## D

`datatype` (*tellcore.telldus.SensorValue* attribute), 7  
`Device` (class in *tellcore.telldus*), 5  
`DeviceFactory()` (in module *tellcore.telldus*), 5  
`DeviceGroup` (class in *tellcore.telldus*), 6  
`devices()` (*tellcore.telldus.TelldusCore* method), 4  
`devices_in_group()` (*tellcore.telldus.DeviceGroup* method), 6  
`dim()` (*tellcore.telldus.Device* method), 5

`DirectCallbackDispatcher` (class in *tellcore.library*), 13  
`disconnect_controller()` (*tellcore.telldus.TelldusCore* method), 4  
`down()` (*tellcore.telldus.Device* method), 5

## E

`execute()` (*tellcore.telldus.Device* method), 5

## F

`from_param()` (*tellcore.library.Library.c\_string\_p* class method), 12

## G

`get_parameter()` (*tellcore.telldus.Device* method), 5

## H

`has_value()` (*tellcore.telldus.Sensor* method), 6

## I

`id` (*tellcore.telldus.Controller* attribute), 7

## L

`last_sent_command()` (*tellcore.telldus.Device* method), 5  
`last_sent_value()` (*tellcore.telldus.Device* method), 5  
`learn()` (*tellcore.telldus.Device* method), 6  
`Library` (class in *tellcore.library*), 11  
`Library.c_string_p` (class in *tellcore.library*), 12

## M

`methods()` (*tellcore.telldus.Device* method), 6  
`model` (*tellcore.telldus.Device* attribute), 5

## N

`name` (*tellcore.telldus.Device* attribute), 5

## O

`on_callback()` (*tellcore.library.BaseCallbackDispatcher method*), 12

`on_callback()` (*tellcore.library.DirectCallbackDispatcher method*), 13

`on_callback()` (*tellcore.telldus.AsyncioCallbackDispatcher method*), 8

`on_callback()` (*tellcore.telldus.QueuedCallbackDispatcher method*), 7

## P

`parameters()` (*tellcore.telldus.Device method*), 6

`process_callback()` (*tellcore.telldus.QueuedCallbackDispatcher method*), 7

`process_pending_callbacks()` (*tellcore.telldus.QueuedCallbackDispatcher method*), 8

`protocol` (*tellcore.telldus.Device attribute*), 5

## Q

`QueuedCallbackDispatcher` (*class in tellcore.telldus*), 7

## R

`register_controller_event()` (*tellcore.telldus.TelldusCore method*), 4

`register_device_change_event()` (*tellcore.telldus.TelldusCore method*), 4

`register_device_event()` (*tellcore.telldus.TelldusCore method*), 4

`register_raw_device_event()` (*tellcore.telldus.TelldusCore method*), 4

`register_sensor_event()` (*tellcore.telldus.TelldusCore method*), 4

`remove()` (*tellcore.telldus.Device method*), 6

`remove_from_group()` (*tellcore.telldus.DeviceGroup method*), 6

## S

`send_raw_command()` (*tellcore.telldus.TelldusCore method*), 4

`Sensor` (*class in tellcore.telldus*), 6

`sensors()` (*tellcore.telldus.TelldusCore method*), 5

`SensorValue` (*class in tellcore.telldus*), 7

`set_parameter()` (*tellcore.telldus.Device method*), 6

`stop()` (*tellcore.telldus.Device method*), 6

## T

`tdController()` (*tellcore.library.Library method*), 12

`tdSensor()` (*tellcore.library.Library method*), 12

`tdSensorValue()` (*tellcore.library.Library method*), 12

`tellcore.constants` (*module*), 9

`tellcore.library` (*module*), 11

`tellcore.telldus` (*module*), 3

`TelldusCore` (*class in tellcore.telldus*), 3

`TelldusError`, 12

`timestamp` (*tellcore.telldus.SensorValue attribute*), 7

`turn_off()` (*tellcore.telldus.Device method*), 6

`turn_on()` (*tellcore.telldus.Device method*), 6

`type` (*tellcore.telldus.Controller attribute*), 7

`type` (*tellcore.telldus.Device attribute*), 5

## U

`unregister_callback()` (*tellcore.telldus.TelldusCore method*), 5

`up()` (*tellcore.telldus.Device method*), 6

## V

`value` (*tellcore.telldus.SensorValue attribute*), 7

`value()` (*tellcore.telldus.Sensor method*), 6