npTDMS Documentation

Release 0.11.0

Adam Reeve

Apr 23, 2017

Contents

1 Contents **3**
1.1 Installation and Quick Start . . . 3
1.2 Reading TDMS files . . . 4
1.3 Writing TDMS files . . . 4
1.4 npTDMS API Reference . . . 6
1.5 The tdmsinfo Command . . . 10
1.6 Limitations . . . 10

Python Module Index **11**

npTDMS is a cross-platform Python package for reading and writing TDMS files as produced by LabVIEW, and is built on top of the numpy package. Data read from a TDMS file is stored in numpy arrays, and numpy arrays are also used when writing TDMS file.

CHAPTER 1

Contents

Installation and Quick Start

npTDMS is available from the Python Package Index, so the easiest way to install it is by running (as root):

```
pip install npTDMS
```

Or you can install npTDMS as a non-root user inside your home directory:

```
pip install --user npTDMS
```

Alternatively, after downloading the source code you can extract it and change into the new directory, then run:

```
python setup.py install
```

Typical usage when reading a TDMS file might look like:

```
from nptdms import TdmsFile

tdms_file = TdmsFile("path_to_file.tdms")
channel = tdms_file.object('Group', 'Channel1')
data = channel.data
# do stuff with data
```

And to write a TDMS file:

```
from nptdms import TdmsWriter, ChannelObject
import numpy

with TdmsWriter("path_to_file.tdms") as tdms_writer:
    data_array = numpy.linspace(0, 1, 10)
    channel = ChannelObject('Group', 'Channel1', data_array)
    tdms_writer.write_segment([channel])
```

Reading TDMS files

To read a TDMS file, create an instance of the *nptdms.TdmsFile* class, passing the path to the file, or an already opened file to the `__init__` method:

```
tdms_file = TdmsFile("my_file.tdms")
```

This will read the contents of the TDMS file, then the various objects in the file can be accessed using the *nptdms.TdmsFile.object()* method. An object in a TDMS file is either the root object, a group object, or a channel object. Only channel objects contain data, but any object may have properties associated with it. For example, it is common to have properties stored against the root object such as the file title and author.

The object returned by the `object` method is an instance of *nptdms.TdmsObject*. If this is a channel containing data, you can access the data as a numpy array using its `data` attribute:

```
channel_object = tdms_file.object("group_name", "channel_name")
data = channel_object.data
```

If the array is waveform data and has the `wf_start_offset` and `wf_increment` properties, you can get an array of relative time values for the data using the *nptdms.TdmsObject.time_track()* method:

```
time = channel_object.time_track()
```

You can access the properties of an object using the *nptdms.TdmsObject.property()* method, or the `nptdms.TdmsObject.properties` dictionary, for example:

```
root_object = tdms_file.object()

# Iterate over all items in the properties dictionary and print them
for name, value in root_object.properties.items():
    print("{0}: {1}".format(name, value))

# Get a single property value
property_value = root_object.property("my_property_name")
```

You may also have group objects in your TDMS files that do not contain channels but are only used to group properties, in which case you can access these objects in the same way as a normal group:

```
attributes = tdms_file.object("attributes").properties
```

Writing TDMS files

npTDMS has rudimentary support for writing TDMS files. The full set of optimisations supported by the TDMS file format for speeding up the writing of files and minimising file size are not implemented by npTDMS, but the basic functionality required to write TDMS files is available.

To write a TDMS file, the *nptdms.TdmsWriter* class is used, which should be used as a context manager. The `__init__` method accepts the path to the file to create, or a file that has already been opened in binary write mode:

```
with TdmsWriter("my_file.tdms") as tdms_writer:
    # write data
```

The *nptdms.TdmsWriter.write_segment()* method is used to write a segment of data to the TDMS file. Because the TDMS file format is designed for streaming data applications, it supports writing data one segment at a

time as data becomes available. If you don't require this functionality you can simple call `write_segment` once with all of your data.

The `write_segment` method takes a list of objects, each of which must be an instance of one of:

- *nptdms.RootObject*. This is the TDMS root object, and there may only be one root object in a segment.
- *nptdms.GroupObject*. This is used to group the channel objects.
- *nptdms.ChannelObject*. An object that contains data.
- *nptdms.TdmsObject*. A TDMS object that was read from a TDMS file using *nptdms.TdmsFile*.

Each of `RootObject`, `GroupObject` and `ChannelObject` may optionally have properties associated with them, which are passed into the `__init__` method as a dictionary. The data types supported as property values are:

- Integers
- Floating point values
- Strings
- datetime objects
- Boolean values

For more control over the data type used to represent a property value, for example to use an unsigned integer type, you can pass an instance of one of the data types from the *nptdms.types* module.

A complete example of writing a TDMS file with various object types and properties is given below:

```
from nptdms import TdmsWriter, RootObject, GroupObject, ChannelObject

root_object = RootObject(properties={
    "prop1": "foo",
    "prop2": 3,
})
group_object = GroupObject("group_1", properties={
    "prop1": 1.2345,
    "prop2": False,
})
data = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
channel_object = ChannelObject("group_1", "channel_1", data, properties={})

with TdmsWriter("my_file.tdms") as tdms_writer:
    tdms_writer.write_segment([
        root_object,
        group_object,
        channel_object])
```

You could also read a TDMS file and then re-write it by passing *nptdms.TdmsObject* instances to the `write_segment` method. If you want to only copy certain objects for example, you could do something like:

```
from nptdms import TdmsFile, TdmsWriter

original_file = TdmsFile("original_file.tdms")

with TdmsWriter("copied_file.tdms") as copied_file:
    objects_to_copy = [obj for obj in original_file.objects.values() if include_
↪object(obj)]
    copied_file.write_segment(objects_to_copy)
```

npTDMS API Reference

Reading TDMS Files

class `nptdms.`**`TdmsFile`**(*file*, *memmap_dir=None*)

Reads and stores data from a TDMS file.

Variables **`objects`** – A dictionary of objects in the TDMS file, where the keys are the object paths.

`__init__`(*file*, *memmap_dir=None*)

Initialise a new TDMS file object, reading all data.

Parameters

- **`file`** – Either the path to the tdms file to read or an already opened file.
- **`memmap_dir`** – The directory to store memmapped data files in, or None to read data into memory. The data files are created as temporary files and are deleted when the channel data is no longer used. tempfile.gettempdir() can be used to get the default temporary file directory.

`as_dataframe`(*time_index=False*, *absolute_time=False*)

Converts the TDMS file to a DataFrame

Parameters

- **`time_index`** – Whether to include a time index for the dataframe.
- **`absolute_time`** – If time_index is true, whether the time index values are absolute times or relative to the start time.

Returns The full TDMS file data.

Return type pandas.DataFrame

`channel_data`(*group*, *channel*)

Get the data for a channel

Parameters

- **`group`** – The name of the group the channel is in.
- **`channel`** – The name of the channel to get data for.

Returns The channel data.

Return type NumPy array.

`group_channels`(*group*)

Returns a list of channel objects for the given group

Parameters **`group`** – Name of the group to get channels for.

Return type List of `TdmsObject` objects.

`groups`()

Return the names of groups in the file

Note that there is not necessarily a TDMS object associated with each group name.

Return type List of strings.

`object`(**path*)

Get a TDMS object from the file

Parameters **path** – The object group and channel. Providing no channel returns a group object, and providing no channel or group will return the root object.

Return type *TdmsObject*

For example, to get the root object:

```
object()
```

To get a group:

```
object("group_name")
```

To get a channel:

```
object("group_name", "channel_name")
```

class nptdms.**TdmsObject**(*path*, *tdms_file=None*)

Represents an object in a TDMS file.

Variables

- ***path*** – The TDMS object path.
- **properties** – Dictionary of TDMS properties defined for this object, for example the start time and time increment for waveforms.
- **has_data** – Boolean, true if there is data associated with the object.

as_dataframe(*absolute_time=False*)

Converts the TDMS object to a DataFrame

Parameters **absolute_time** – Whether times should be absolute rather than relative to the start time.

Returns The TDMS object data.

Return type pandas.DataFrame

channel

Returns the name of the channel for this object, or None if it is a group or the root object.

data

NumPy array containing data if there is data for this object, otherwise None.

group

Returns the name of the group for this object, or None if it is the root object.

property(*property_name*)

Returns the value of a TDMS property

Parameters **property_name** – The name of the property to get.

Returns The value of the requested property.

Raises KeyError if the property isn't found.

raw_data

For objects that contain DAQmx raw data, this is the raw, unscaled data array. For other objects this is the same as the data property.

time_track(*absolute_time=False*, *accuracy='ns'*)

Return an array of time or the independent variable for this channel

This depends on the object having the wf_increment and wf_start_offset properties defined. Note that wf_start_offset is usually zero for time-series data. If you have time-series data channels with different start times, you should use the absolute time or calculate the time offsets using the wf_start_time property.

For larger timespans, the accuracy setting should be set lower. The default setting is 'ns', which has a timespan of [1678 AD, 2262 AD]. For the exact ranges, refer to http://docs.scipy.org/doc/numpy/reference/arrays.datetime.html section "Datetime Units".

Parameters

- **absolute_time** – Whether the returned time values are absolute times rather than relative to the start time. If true, the wf_start_time property must be set.
- **accuracy** – The accuracy of the returned datetime64 array.

Return type NumPy array.

Raises KeyError if required properties aren't found

Writing TDMS Files

class nptdms.**TdmsWriter**(*file*, *mode='w'*)

Writes to a TDMS file.

A TdmsWriter should be used as a context manager, for example:

```
with TdmsWriter(path) as tdms_writer:
    tdms_writer.write_segment(segment_data)
```

__init__(*file*, *mode='w'*)

Initialise a new TDMS writer

Parameters

- **file** – Either the path to the tdms file to open or an already opened file.
- **mode** – Either 'w' to open a new file or 'a' to append to an existing TDMS file.

write_segment(*objects*)

Write a segment of data to a TDMS file

Parameters **objects** – A list of TdmsObject instances to write

class nptdms.**RootObject**(*properties=None*)

The root TDMS object

__init__(*properties=None*)

Initialise a new GroupObject

Parameters **properties** – A dictionary mapping property names to their value.

path

The string representation of the root path

class nptdms.**GroupObject**(*group*, *properties=None*)

A TDMS object for a group

__init__(*group*, *properties=None*)

Initialise a new GroupObject

Parameters

- **group** – The name of this group.

- **properties** – A dictionary mapping property names to their value.

path
The string representation of this group's path

class nptdms.**ChannelObject**(*group*, *channel*, *data*, *properties=None*)
A TDMS object for a channel with data

__init__(*group*, *channel*, *data*, *properties=None*)
Initialise a new ChannelObject

Parameters

- **group** – The name of the group this channel is in.
- **channel** – The name of this channel.
- **data** – 1-D Numpy array of data to be written.
- **properties** – A dictionary mapping property names to their value.

path
The string representation of this channel's path

Data Types for Property Values

class nptdms.types.**Int8**(*value*)

class nptdms.types.**Int16**(*value*)

class nptdms.types.**Int32**(*value*)

class nptdms.types.**Int64**(*value*)

class nptdms.types.**Uint8**(*value*)

class nptdms.types.**Uint16**(*value*)

class nptdms.types.**Uint32**(*value*)

class nptdms.types.**Uint64**(*value*)

class nptdms.types.**SingleFloat**(*value*)

class nptdms.types.**DoubleFloat**(*value*)

class nptdms.types.**String**(*value*)

class nptdms.types.**Boolean**(*value*)

class nptdms.types.**TimeStamp**(*value*)

Indices and Tables

- genindex
- modindex
- search

The tdmsinfo Command

npTDMS comes with a command line program, `tdmsinfo`, which lists the contents of a TDMS file. Usage looks like:

```
tdmsinfo [--properties] tdms_file
```

Passing the `--properties` or `-p` argument will include TDMS object properties in the printed information.

The output of tdmsinfo including properties will look something like:

```
/
properties:
name: test_file
  /'group_1'
  properties:
  group_property: property value
    /'group_1'/'channel_a'
      data type: tdsTypeU8
      properties:
      wf_start_time: 2016-12-3014:56:00+00:00
      wf_increment: 0.0005
      wf_samples: 2000
```

There is also a `--debug` or `-d` argument that will output debug information to stderr, which can be useful when debugging a problem with a TDMS file.

Limitations

npTDMS currently doesn't support reading TDMS files with XML headers, or files with extended floating point data.

Python Module Index

n

nptdms, 6
nptdms.types, 9

Index

Symbols

__init__() (nptdms.ChannelObject method), 9
__init__() (nptdms.GroupObject method), 8
__init__() (nptdms.RootObject method), 8
__init__() (nptdms.TdmsFile method), 6
__init__() (nptdms.TdmsWriter method), 8

A

as_dataframe() (nptdms.TdmsFile method), 6
as_dataframe() (nptdms.TdmsObject method), 7

B

Boolean (class in nptdms.types), 9

C

channel (nptdms.TdmsObject attribute), 7
channel_data() (nptdms.TdmsFile method), 6
ChannelObject (class in nptdms), 9

D

data (nptdms.TdmsObject attribute), 7
DoubleFloat (class in nptdms.types), 9

G

group (nptdms.TdmsObject attribute), 7
group_channels() (nptdms.TdmsFile method), 6
GroupObject (class in nptdms), 8
groups() (nptdms.TdmsFile method), 6

I

Int16 (class in nptdms.types), 9
Int32 (class in nptdms.types), 9
Int64 (class in nptdms.types), 9
Int8 (class in nptdms.types), 9

N

nptdms (module), 6
nptdms.types (module), 9

O

object() (nptdms.TdmsFile method), 6

P

path (nptdms.ChannelObject attribute), 9
path (nptdms.GroupObject attribute), 9
path (nptdms.RootObject attribute), 8
property() (nptdms.TdmsObject method), 7

R

raw_data (nptdms.TdmsObject attribute), 7
RootObject (class in nptdms), 8

S

SingleFloat (class in nptdms.types), 9
String (class in nptdms.types), 9

T

TdmsFile (class in nptdms), 6
TdmsObject (class in nptdms), 7
TdmsWriter (class in nptdms), 8
time_track() (nptdms.TdmsObject method), 7
TimeStamp (class in nptdms.types), 9

U

Uint16 (class in nptdms.types), 9
Uint32 (class in nptdms.types), 9
Uint64 (class in nptdms.types), 9
Uint8 (class in nptdms.types), 9

W

write_segment() (nptdms.TdmsWriter method), 8