RobotPy networktables Documentation

Release 2020.0.4

RobotPy Development Team

Apr 04, 2020

ROBOT PROGRAMMING

This is a pure python implementation of the NetworkTables protocol, derived from the wpilib ntcore C++ implementation. In FRC, the NetworkTables protocol is used to pass non-Driver Station data to and from the robot across the network.

Don't understand this NetworkTables thing? Check out our basic overview of NetworkTables.

This implementation is intended to be compatible with python 2.7 and python 3.3+. All commits to the repository are automatically tested on all supported python versions using Travis-CI.

Note: NetworkTables is a protocol used for robot communication in the FIRST Robotics Competition, and can be used to talk to SmartDashboard/SFX. It does not have any security, and should never be used on untrusted networks.

CHAPTER

ONE

API REFERENCE

1.1 Examples

These are the simple examples that are included with pynetworktables.

1.1.1 Robot Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables server (eg, the robot or simulator side).
#
# On a real robot, you probably would create an instance of the
# wpilib.SmartDashboard object and use that instead -- but it's really
# just a passthru to the underlying NetworkTable object.
#
# When running, this will continue incrementing the value 'robotTime',
# and the value should be visible to networktables clients such as
# SmartDashboard. To view using the SmartDashboard, you can launch it
# like so:
#
#     SmartDashboard.jar ip 127.0.0.1
#

import time
from networktables import NetworkTables

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

NetworkTables.initialize()
sd = NetworkTables.getTable("SmartDashboard")

i = 0
while True:
    print("dsTime:", sd.getNumber("dsTime", -1))

    sd.putNumber("robotTime", i)
    time.sleep(1)
    i += 1
```

1.1.2 Driver Station Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# When running, this will continue incrementing the value 'dsTime', and the
# value should be visible to other networktables clients and the robot.
#

import sys
import time
from networktables import NetworkTables

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)

sd = NetworkTables.getTable("SmartDashboard")

i = 0
while True:
    print("robotTime:", sd.getNumber("robotTime", -1))

    sd.putNumber("dsTime", i)
    time.sleep(1)
    i += 1
```

1.1.3 Listener Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# This shows how to use a listener to listen for changes in NetworkTables
# values. This will print out any changes detected on the SmartDashboard
# table.
#

import sys
import time
from networktables import NetworkTables
```

(continues on next page)

(continued from previous page)

```
# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)


def valueChanged(table, key, value, isNew):
    print("valueChanged: key: '%s'; value: %s; isNew: %s" % (key, value, isNew))


def connectionListener(connected, info):
    print(info, "; Connected=%s" % connected)


NetworkTables.addConnectionListener(connectionListener, immediateNotify=True)

sd = NetworkTables.getTable("SmartDashboard")
sd.addEntryListener(valueChanged)

while True:
    time.sleep(1)
```

1.1.4 Listen Chooser Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# This shows how to use a listener to listen for changes to a SendableChooser
# object.
#

from __future__ import print_function

import sys
import time
from networktables import NetworkTables
from networktables.util import ChooserControl

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)
```

(continues on next page)

(continued from previous page)

```
if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)


def on_choices(value):
    print("OnChoices", value)


def on_selected(value):
    print("OnSelected", value)


cc = ChooserControl("Autonomous Mode", on_choices, on_selected)

while True:
    time.sleep(1)
```

1.1.5 Auto Listener Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# When running, this will create an automatically updated value, and print
# out the value.
#

import sys
import time
from networktables import NetworkTables

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)

sd = NetworkTables.getTable("SmartDashboard")
auto_value = sd.getAutoUpdateValue("robotTime", 0)

while True:
```

(continues on next page)

(continued from previous page)

```
    print("robotTime:", auto_value.value)
    time.sleep(1)
```

1.1.6 Global Listener Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# This shows how to use a listener to listen for all changes in NetworkTables
# values, which prints out all changes. Note that the keys are full paths, and
# not just individual key values.
#

import sys
import time
from networktables import NetworkTables

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)


def valueChanged(key, value, isNew):
    print("valueChanged: key: '%s'; value: %s; isNew: %s" % (key, value, isNew))


NetworkTables.addEntryListener(valueChanged)

while True:
    time.sleep(1)
```

1.1.7 ntproperty Example

```
#!/usr/bin/env python3
#
# This is a NetworkTables client (eg, the DriverStation/coprocessor side).
# You need to tell it the IP address of the NetworkTables server (the
# robot or simulator).
#
# When running, this will continue incrementing the value 'dsTime', and the
```

(continues on next page)

(continued from previous page)

```
# value should be visible to other networktables clients and the robot.
#

import sys
import time
from networktables import NetworkTables
from networktables.util import ntproperty

# To see messages from networktables, you must setup logging
import logging

logging.basicConfig(level=logging.DEBUG)

if len(sys.argv) != 2:
    print("Error: specify an IP to connect to!")
    exit(0)

ip = sys.argv[1]

NetworkTables.initialize(server=ip)


class SomeClient(object):
    """Demonstrates an object with magic networktables properties"""

    robotTime = ntproperty("/SmartDashboard/robotTime", 0, writeDefault=False)

    dsTime = ntproperty("/SmartDashboard/dsTime", 0)


c = SomeClient()


i = 0
while True:

    # equivalent to wpilib.SmartDashboard.getNumber('robotTime', None)
    print("robotTime:", c.robotTime)

    # equivalent to wpilib.SmartDashboard.putNumber('dsTime', i)
    c.dsTime = i

    time.sleep(1)
    i += 1
```

1.1.8 json_logger Example

This is a more complex example which can be used to log data from your robot into a JSON file. There is a corresponding

As this example is a bit larger than the others, see the 'samples/json_logger' directory of the pynetworktables repository on github. It also includes a script that you can modify to plot the data.

1.2 NetworkTables API

class networktables.**NetworkTablesInstance**

The object `networktables.NetworkTables` is a global singleton that you can use to initialize NetworkTables connections, configure global settings and listeners, and to create table objects which can be used to send data to/from NetworkTable servers and clients.

First, you must initialize NetworkTables:

```
from networktables import NetworkTables

# As a client to connect to a robot
NetworkTables.initialize(server='roborio-XXX-frc.local')
```

Then, to interact with the SmartDashboard you get an instance of the table, and you can call the various methods:

```
sd = NetworkTables.getTable('SmartDashboard')

sd.putNumber('someNumber', 1234)
otherNumber = sd.getNumber('otherNumber')
```

You can create additional NetworkTablesInstance objects. Instances are completely independent from each other. Table operations on one instance will not be visible to other instances unless the instances are connected via the network. The main limitation on instances is that you cannot have two servers on the same network port. The main utility of instances is for unit testing, but they can also enable one program to connect to two different NetworkTables networks.

The global "default" instance (as returned by *NetworkTablesInstance.getDefault()*) is always available, and is intended for the common case when there is only a single NetworkTables instance being used in the program.

Additional instances can be created with the *create()* function.

See also:

- The examples in the documentation.
- *NetworkTable*

DEFAULT_PORT = 1735

The default port that network tables operates on

class EntryFlags

NetworkTables entry flags

PERSISTENT = 1

Indicates a value that will be persisted on the server

class EntryTypes

NetworkTable value types used in *NetworkTable.getKeys()*

BOOLEAN = b'\x01'

True or False

BOOLEAN_ARRAY = b'\x10'

List of booleans

DOUBLE = b'\x02'

Floating point number

DOUBLE_ARRAY = b' '
List of numbers

RAW = b'\x08'
Raw bytes

STRING = b'\x04'
Strings

STRING_ARRAY = b'@'
List of strings

class NetworkModes
Bitflags returend from *getNetworkMode()*

CLIENT = 2
Running in client mode

FAILURE = 8
Flag for failure (either client or server)

NONE = 0
Not running

SERVER = 1
Running in server mode

STARTING = 4
Flag for starting (either client or server)

TEST = 16
Flag indicating in test mode

class NotifyFlags
Bitflags passed to entry callbacks

DELETE = 8
Key deleted

FLAGS = 32
Flags changed

IMMEDIATE = 1
Initial listener addition

LOCAL = 2
Changed locally

NEW = 4
Newly created entry

UPDATE = 16
Value changed

PATH_SEPARATOR = '/'
The path separator for sub-tables and keys

addConnectionListener(*listener*, *immediateNotify=False*)
Adds a listener that will be notified when a new connection to a NetworkTables client/server is established.

The listener is called from a NetworkTables owned thread and should return as quickly as possible.

Parameters

- **listener** (*fn(bool, ConnectionInfo)*) – A function that will be called with two parameters
- **immediateNotify** (bool) – If True, the listener will be called immediately with any active connection information

> **Warning:** You may call the NetworkTables API from within the listener, but it is not recommended.

Changed in version 2017.0.0: The listener is now a function

addEntryListener(*listener*, *immediateNotify=True*, *localNotify=True*, *paramIsNew=True*)

Adds a listener that will be notified when any key in any NetworkTable is changed. The keys that are received using this listener will be full NetworkTable keys. Most users will not want to use this listener type.

The listener is called from the NetworkTables I/O thread, and should return as quickly as possible.

Parameters

- **listener** (Callable[[str, Any, int], None]) – A callable that has this signature: *callable(key, value, isNew)*
- **immediateNotify** (bool) – If True, the listener will be called immediately with the current values of the table
- **localNotify** (bool) – True if you wish to be notified of changes made locally (default is True)
- **paramIsNew** (bool) – If True, the listener third parameter is a boolean set to True if the listener is being called because of a new value in the table. Otherwise, the parameter is an integer of the raw *NT_NOTIFY_** flags

New in version 2015.2.0.

Changed in version 2017.0.0: *paramIsNew* parameter added

Changed in version 2018.0.0: Renamed to addEntryListener, no longer initializes NetworkTables

> **Warning:** You may call the NetworkTables API from within the listener, but it is not recommended as we are not currently sure if deadlocks will occur

Return type None

addEntryListenerEx(*listener*, *flags*, *paramIsNew=True*)

Adds a listener that will be notified when any key in any NetworkTable is changed. The keys that are received using this listener will be full NetworkTable keys. Most users will not want to use this listener type.

The listener is called from the NetworkTables I/O thread, and should return as quickly as possible.

Parameters

- **listener** (Callable[[str, Any, int], None]) – A callable that has this signature: *callable(key, value, isNew)*
- **flags** (*NotifyFlags*) – Bitmask of flags that indicate the types of notifications you wish to receive

- **paramIsNew** (bool) – If True, the listener third parameter is a boolean set to True if the listener is being called because of a new value in the table. Otherwise, the parameter is an integer of the raw *NT_NOTIFY_** flags

New in version 2017.0.0.

Changed in version 2018.0.0: Renamed to addEntryListenerEx, no longer initializes NetworkTables

Return type None

classmethod create()
Create an instance.

Return type NetworkTablesInstance

Returns Newly created instance

deleteAllEntries()
Deletes ALL keys in ALL subtables (except persistent values). Use with caution!

New in version 2018.0.0.

Return type None

enableVerboseLogging()
Enable verbose logging that can be useful when trying to diagnose NetworkTables issues.

Warning: Don't enable this in normal use, as it can potentially cause performance issues due to the volume of logging.

New in version 2017.0.0.

Return type None

flush()
Flushes all updated values immediately to the network.

Note: This is rate-limited to protect the network from flooding. This is primarily useful for synchronizing network updates with user code.

New in version 2017.0.0.

Return type None

getConnections()
Gets information on the currently established network connections. If operating as a client, this will return either zero or one values.

Returns list of connection information

Return type list

New in version 2018.0.0.

classmethod getDefault()
Get global default instance.

Return type NetworkTablesInstance

getEntries(*prefix*, *types=0*)
Get entries starting with the given prefix. The results are optionally filtered by string prefix and entry type to only return a subset of all entries.

Parameters

- **prefix** (str) – entry name required prefix; only entries whose name starts with this string are returned
- **types** (int) – bitmask of types; 0 is treated as a "don't care"

Returns List of matching entries.

Return type list of *NetworkTableEntry*

New in version 2018.0.0.

getEntry(*name*)

Gets the entry for a key.

Parameters **name** (str) – Absolute path of key

Return type NetworkTableEntry

Returns Network table entry.

New in version 2018.0.0.

getEntryInfo(*prefix*, *types=0*)

Get information about entries starting with the given prefix. The results are optionally filtered by string prefix and entry type to only return a subset of all entries.

Parameters

- **prefix** (str) – entry name required prefix; only entries whose name starts with this string are returned
- **types** (int) – bitmask of types; 0 is treated as a "don't care"

Return type Sequence

Returns List of entry information.

New in version 2018.0.0.

getGlobalAutoUpdateValue(*key*, *defaultValue*, *writeDefault*)

Global version of getAutoUpdateValue.

Parameters

- **key** (str) – the full NT path of the value (must start with /)
- **defaultValue** – The default value to return if the key doesn't exist
- **writeDefault** (bool) – If True, force the value to the specified default

Return type *NetworkTableEntry*

See also:

ntproperty() is a read-write alternative to this

New in version 2015.3.0.

Changed in version 2018.0.0: This now returns the same as *NetworkTablesInstance.getEntry()*

getGlobalTable()

Returns an object that allows you to write values to absolute NetworkTable keys (which are paths with / separators).

Note: This is now an alias for `NetworkTables.getTable('/')`

New in version 2015.2.0.

Changed in version 2017.0.0: Returns a NetworkTable instance

Changed in version 2018.0.0: No longer automatically initializes network tables

Return type `NetworkTable`

getNetworkMode()
Get the current network mode

New in version 2018.0.0.

getRemoteAddress()
Only returns a valid address if connected to the server. If this is a server, returns None

Return type `Optional[str]`

Returns IP address of server or None

New in version 2015.3.2.

getTable(*key*)
Gets the table with the specified key.

Parameters **key** (`str`) – the key name

Return type `NetworkTable`

Returns the network table requested

Changed in version 2018.0.0: No longer automatically initializes network tables

initialize(*server=None*)
Initializes NetworkTables and begins operations

Parameters **server** (*str*) – If specified, NetworkTables will be set to client mode and attempt to connect to the specified server. This is equivalent to executing:

```
self.startClient(server)
```

Returns True if initialized, False if already initialized

New in version 2017.0.0: The *server* parameter

isConnected()

Return type `bool`

Returns True if connected to at least one other NetworkTables instance

isServer()

Return type `bool`

Returns True if configured in server mode

loadEntries(*filename*, *prefix*)
Load table values from a file. The file format used is identical to that used for SavePersistent / LoadPersistent.

Parameters

- **filename** (`str`) – filename

- **prefix** (str) – load only keys starting with this prefix

Returns None if success, or a string describing the error on failure

New in version 2018.0.0.

loadPersistent(*filename*)
Loads persistent keys from a file. WPILib will do this automatically on a robot server.

Parameters **filename** (str) – Name of file to load keys from

Returns None if success, or a string describing the error on failure

New in version 2017.0.0.

removeConnectionListener(*listener*)
Removes a connection listener

Parameters **listener** (Callable) – The function registered for connection notifications

removeEntryListener(*listener*)
Remove an entry listener.

Parameters **listener** (Callable[[str, Any, int], None]) – Listener to remove

New in version 2018.0.0.

Return type None

saveEntries(*filename*, *prefix*)
Save table values to a file. The file format used is identical to that used for SavePersistent.

Parameters

- **filename** (str) – filename
- **prefix** (str) – save only keys starting with this prefix

Returns None if success, or a string describing the error on failure

New in version 2018.0.0.

savePersistent(*filename*)
Saves persistent keys to a file. The server does this automatically.

Parameters **filename** (str) – Name of file to save keys to

Returns None if success, or a string describing the error on failure

New in version 2017.0.0.

setDashboardMode(*port=1735*)
Starts requesting server address from Driver Station. This connects to the Driver Station running on localhost to obtain the server IP address.

Parameters **port** (int) – server port to use in combination with IP from DS

New in version 2018.0.0: Was formerly called setDashboardMode

Return type None

setNetworkIdentity(*name*)
Sets the network identity of this node. This is the name used in the initial connection handshake, and is provided in the connection info on the remote end.

Parameters **name** (str) – A string to communicate to other NetworkTables instances

New in version 2017.0.0.

Return type None

setServer(*server_or_servers*)

Sets server addresses and port for client (without restarting client). The client will attempt to connect to each server in round robin fashion.

Parameters **server_or_servers** (Union[str, Tuple[str, int], List[Tuple[str, int]], List[str]]) – a string, a tuple of (server, port), array of (server, port), or an array of strings

New in version 2018.0.0.

Return type None

setServerTeam(*team*, *port=1735*)

Sets server addresses and port for client based on the team number (without restarting client). The client will attempt to connect to each server in round robin fashion.

Parameters

- **team** (int) – Team number
- **port** (int) – Port to communicate over

New in version 2018.0.0.

Return type None

setUpdateRate(*interval*)

Sets the period of time between writes to the network.

WPILib's networktables and SmartDashboard default to 100ms, we have set it to 50ms instead for quicker response time. You should not set this value too low, as it could potentially increase the volume of data sent over the network.

Parameters **interval** (float) – Write flush period in seconds (default is 0.050, or 50ms)

> **Warning:** If you don't know what this setting affects, don't mess with it!

New in version 2017.0.0.

Return type None

shutdown()

Stops all NetworkTables activities and unregisters all tables and callbacks. You can call *initialize()* again after calling this.

New in version 2017.0.0.

Return type None

startClient(*server_or_servers*)

Sets server addresses and port for client (without restarting client). The client will attempt to connect to each server in round robin fashion.

Parameters **server_or_servers** (Union[str, Tuple[str, int], List[Tuple[str, int]], List[str]]) – a string, a tuple of (server, port), array of (server, port), or an array of strings

New in version 2018.0.0.

startClientTeam(*team*, *port=1735*)

Starts a client using commonly known robot addresses for the specified team.

Parameters

- **team** (int) – team number
- **port** (int) – port to communicate over

New in version 2018.0.0.

startDSClient(*port=1735*)

Starts requesting server address from Driver Station. This connects to the Driver Station running on localhost to obtain the server IP address.

Parameters **port** (int) – server port to use in combination with IP from DS

New in version 2018.0.0: Was formerly called setDashboardMode

Return type None

startServer(*persistFilename='networktables.ini'*, *listenAddress=''*, *port=1735*)

Starts a server using the specified filename, listening address, and port.

Parameters

- **persistFilename** (str) – the name of the persist file to use
- **listenAddress** (str) – the address to listen on, or empty to listen on any address
- **port** (int) – port to communicate over

New in version 2018.0.0.

startTestMode(*server=True*)

Setup network tables to run in unit test mode, and enables verbose logging.

Returns True if successful

New in version 2018.0.0.

stopClient()

Stops the client if it is running.

New in version 2018.0.0.

Return type None

stopServer()

Stops the server if it is running.

New in version 2018.0.0.

Return type None

waitForConnectionListenerQueue(*timeout*)

Wait for the connection listener queue to be empty. This is primarily useful for deterministic testing. This blocks until either the connection listener queue is empty (e.g. there are no more events that need to be passed along to callbacks or poll queues) or the timeout expires.

Parameters **timeout** (float) – timeout, in seconds. Set to 0 for non-blocking behavior, or a negative value to block indefinitely

Return type bool

Returns False if timed out, otherwise true.

New in version 2018.0.0.

waitForEntryListenerQueue(*timeout*)
Wait for the entry listener queue to be empty. This is primarily useful for deterministic testing. This blocks until either the entry listener queue is empty (e.g. there are no more events that need to be passed along to callbacks or poll queues) or the timeout expires.

> **Warning:** This function is not efficient, so only use it for testing!

Parameters **timeout** (`float`) – timeout, in seconds. Set to 0 for non-blocking behavior, or None to block indefinitely

Return type `bool`

Returns False if timed out, otherwise true.

1.3 NetworkTable Objects

class networktables.**NetworkTable**(*path*, *api*, *inst*)
This is a NetworkTable object, it allows you to interact with NetworkTables in a table-based manner. You should not directly create a NetworkTable object, but instead use `NetworkTables.getTable()` to retrieve a NetworkTable instance.

For example, to interact with the SmartDashboard:

```
from networktables import NetworkTables
sd = NetworkTables.getTable('SmartDashboard')

someNumberEntry = sd.getEntry('someNumber')
someNumberEntry.putNumber(1234)
...
```

See also:

- The examples in the documentation.
- *NetworkTablesInstance*

PATH_SEPARATOR = '/'

addEntryListener(*listener*, *immediateNotify=False*, *key=None*, *localNotify=False*)
Adds a listener that will be notified when any key in this NetworkTable is changed, or when a specified key changes.

The listener is called from the NetworkTables I/O thread, and should return as quickly as possible.

Parameters

- **listener** (`Callable`) – A callable with signature *callable(source, key, value, isNew)*
- **immediateNotify** (`bool`) – If True, the listener will be called immediately with the current values of the table
- **key** (`Optional[str]`) – If specified, the listener will only be called when this key is changed
- **localNotify** (`bool`) – True if you wish to be notified of changes made locally (default is False)

> **Warning:** You may call the NetworkTables API from within the listener, but it is not recommended

Changed in version 2017.0.0: Added localNotify parameter (defaults to False, which is different from NT2)

Return type None

addEntryListenerEx(*listener*, *flags*, *key=None*, *paramIsNew=True*)

Adds a listener that will be notified when any key in this NetworkTable is changed, or when a specified key changes.

The listener is called from the NetworkTables I/O thread, and should return as quickly as possible.

Parameters

- **listener** (Callable) – A callable with signature *callable(source, key, value, param)*
- **flags** (*NotifyFlags*) – Bitmask of flags that indicate the types of notifications you wish to receive
- **key** (Optional[str]) – If specified, the listener will only be called when this key is changed
- **paramIsNew** (bool) – If True, the listener fourth parameter is a boolean set to True if the listener is being called because of a new value in the table. Otherwise, the parameter is an integer of the raw *NT_NOTIFY_** flags

> **Warning:** You may call the NetworkTables API from within the listener, but it is not recommended

New in version 2017.0.0.

Return type None

addSubTableListener(*listener*, *localNotify=False*)

Adds a listener that will be notified when any key in a subtable of this NetworkTable is changed.

The listener is called from the NetworkTables I/O thread, and should return as quickly as possible.

Parameters

- **listener** (Callable) – Callable to call when previously unseen table appears. Function signature is *callable(source, key, subtable, True)*
- **localNotify** (bool) – True if you wish to be notified when local changes result in a new table

> **Warning:** You may call the NetworkTables API from within the listener, but it is not recommended as we are not currently sure if deadlocks will occur

Changed in version 2017.0.0: Added localNotify parameter

Return type None

clearFlags(*key*, *flags*)

Clears entry flags on the specified key in this table.

Parameters

- **key** (str) – the key name

- **flags** (*EntryFlags*) – the flags to clear (bitmask)

New in version 2017.0.0.

Return type None

clearPersistent(*key*)
Stop making a key's value persistent through program restarts.

Parameters **key** (str) – the key name

New in version 2017.0.0.

Return type None

containsKey(*key*)
Determines whether the given key is in this table.

Parameters **key** (str) – the key to search for

Return type bool

Returns True if the table as a value assigned to the given key

containsSubTable(*key*)
Determines whether there exists a non-empty subtable for this key in this table.

Parameters **key** (str) – the key to search for (must not end with path separator)

Return type bool

Returns True if there is a subtable with the key which contains at least one key/subtable of its own

delete(*key*)
Deletes the specified key in this table.

Parameters **key** (str) – the key name

New in version 2017.0.0.

Return type None

getAutoUpdateValue(*key*, *defaultValue*, *writeDefault=True*)
Returns an object that will be automatically updated when the value is updated by networktables.

Parameters

- **key** (str) – the key name
- **defaultValue** (*any*) – Default value to use if not in the table
- **writeDefault** (bool) – If True, put the default value to the table, overwriting existing values

Return type *NetworkTableEntry*

Note: If you modify the returned value, the value will NOT be written back to NetworkTables (though now there are functions you can use to write values). See *ntproperty()* if you're looking for that sort of thing.

See also:

ntproperty() is a better alternative to use

New in version 2015.1.3.

Changed in version 2018.0.0: This now returns the same as *NetworkTable.getEntry()*

getBoolean(*key*, *defaultValue*)
Gets the boolean associated with the given name. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (str) – the key name
- **defaultValue** (bool) – the default value if no value is found

Return type bool

Returns the key

getBooleanArray(*key*, *defaultValue*)
Returns the boolean array the key maps to. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (*str*) – the key to look up
- **defaultValue** – the value to be returned if no value is found

Return type Sequence[bool]

Returns the value associated with the given key or the given default value if there is no value associated with the key

New in version 2017.0.0.

getEntry(*key*)
Gets the entry for a subkey. This is the preferred API to use to access NetworkTable keys.

Return type *NetworkTableEntry*

New in version 2018.0.0.

getFlags(*key*)
Returns the entry flags for the specified key.

Parameters **key** (str) – the key name

Returns the flags, or 0 if the key is not defined

Return type *EntryFlags*

New in version 2017.0.0.

getKeys(*types=0*)

Parameters **types** (*EntryTypes*) – bitmask of types; 0 is treated as a "don't care".

Return type List[str]

Returns keys currently in the table

New in version 2017.0.0.

getNumber(*key*, *defaultValue*)
Gets the number associated with the given name.

Parameters

- **key** (str) – the key to look up
- **defaultValue** (float) – the value to be returned if no value is found

Return type `float`

Returns the value associated with the given key or the given default value if there is no value associated with the key

getNumberArray(*key*, *defaultValue*)

Returns the number array the key maps to. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (`str`) – the key to look up
- **defaultValue** – the value to be returned if no value is found

Return type `Sequence[float]`

Returns the value associated with the given key or the given default value if there is no value associated with the key

New in version 2017.0.0.

getPath()

Gets the full path of this table. Does not include the trailing "/".

Return type `str`

Returns The path (e.g "", "/foo").

getRaw(*key*, *defaultValue*)

Returns the raw value (byte array) the key maps to. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (`str`) – the key to look up
- **defaultValue** (`bytes`) – the value to be returned if no value is found

Return type `bytes`

Returns the value associated with the given key or the given default value if there is no value associated with the key

New in version 2017.0.0.

getString(*key*, *defaultValue*)

Gets the string associated with the given name. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (`str`) – the key to look up
- **defaultValue** (`str`) – the value to be returned if no value is found

Return type `str`

Returns the value associated with the given key or the given default value if there is no value associated with the key

getStringArray(*key*, *defaultValue*)

Returns the string array the key maps to. If the key does not exist or is of different type, it will return the default value.

Parameters

- **key** (str) – the key to look up
- **defaultValue** – the value to be returned if no value is found

Return type Sequence[str]

Returns the value associated with the given key or the given default value if there is no value associated with the key

New in version 2017.0.0.

getSubTable(*key*)
Returns the table at the specified key. If there is no table at the specified key, it will create a new table

Parameters **key** (str) – the key name

Return type NetworkTable

Returns the networktable to be returned

getSubTables()

Return type List[str]

Returns subtables currently in the table

New in version 2017.0.0.

getValue(*key*, *defaultValue*)
Gets the value associated with a key. This supports all NetworkTables types (unlike *putValue()*).

Parameters

- **key** (str) – the key of the value to look up
- **defaultValue** (*any*) – The default value to return if the key doesn't exist

Returns the value associated with the given key

Return type bool, int, float, str, bytes, tuple

New in version 2017.0.0.

isPersistent(*key*)
Returns whether the value is persistent through program restarts.

Parameters **key** (str) – the key name

New in version 2017.0.0.

Return type bool

path = None
Path of table without trailing slash

putBoolean(*key*, *value*)
Put a boolean in the table

Parameters

- **key** (str) – the key to be assigned to
- **value** (bool) – the value that will be assigned

Return type bool

Returns False if the table key already exists with a different type

putBooleanArray(*key*, *value*)
Put a boolean array in the table

Parameters

- **key** (`str`) – the key to be assigned to
- **value** (`Sequence[bool]`) – the value that will be assigned

Return type `bool`

Returns False if the table key already exists with a different type

New in version 2017.0.0.

putNumber(*key*, *value*)
Put a number in the table

Parameters

- **key** (`str`) – the key to be assigned to
- **value** (`float`) – the value that will be assigned

Return type `bool`

Returns False if the table key already exists with a different type

putNumberArray(*key*, *value*)
Put a number array in the table

Parameters

- **key** (`str`) – the key to be assigned to
- **value** (`Sequence[float]`) – the value that will be assigned

Return type `bool`

Returns False if the table key already exists with a different type

New in version 2017.0.0.

putRaw(*key*, *value*)
Put a raw value (byte array) in the table

Parameters

- **key** (`str`) – the key to be assigned to
- **value** (`bytes`) – the value that will be assigned

Return type `bool`

Returns False if the table key already exists with a different type

New in version 2017.0.0.

putString(*key*, *value*)
Put a string in the table

Parameters

- **key** (`str`) – the key to be assigned to
- **value** (`str`) – the value that will be assigned

Return type `bool`

Returns False if the table key already exists with a different type

putStringArray(*key*, *value*)
Put a string array in the table

Parameters

- **key** (str) – the key to be assigned to
- **value** (Sequence[str]) – the value that will be assigned

Return type bool

Returns False if the table key already exists with a different type

New in version 2017.0.0.

putValue(*key*, *value*)
Put a value in the table, trying to autodetect the NT type of the value. Refer to this table to determine the type mapping:

PyType	NT Type	Notes
bool	*EntryTypes.BOOLEAN*	
int	*EntryTypes.DOUBLE*	
float	*EntryTypes.DOUBLE*	
str	*EntryTypes.STRING*	
bytes	*EntryTypes.RAW*	
list	Error	Use *putXXXArray* methods instead
tuple	Error	Use *putXXXArray* methods instead

Parameters

- **key** (str) – the key to be assigned to
- **value** (*bool, int, float, str, bytes*) – the value that will be assigned

Return type bool

Returns False if the table key already exists with a different type

New in version 2017.0.0.

removeEntryListener(*listener*)
Removes a table listener

Parameters **listener** (Callable) – callable that was passed to addTableListener() or *addSubTableListener()*

Return type None

setDefaultBoolean(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (bool) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultBooleanArray(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (Sequence[bool]) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultNumber(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (*int, float*) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultNumberArray(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (Sequence[float]) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultRaw(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (bytes) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultString(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (str) – the key to be assigned to
- **defaultValue** (str) – the default value to set if key doesn't exist.

Return type bool

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultStringArray(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (`str`) – the key to be assigned to
- **defaultValue** (`Sequence[str]`) – the default value to set if key doesn't exist.

Return type `bool`

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultValue(*key*, *defaultValue*)
If the key doesn't currently exist, then the specified value will be assigned to the key.

Parameters

- **key** (`str`) – the key to be assigned to
- **defaultValue** (*bool, int, float, str, bytes*) – the default value to set if key doesn't exist.

Return type `bool`

Returns False if the table key exists with a different type

New in version 2017.0.0.

See also:

putValue()

setFlags(*key*, *flags*)
Sets entry flags on the specified key in this table.

Parameters

- **key** (`str`) – the key name
- **flags** (*EntryFlags*) – the flags to set (bitmask)

New in version 2017.0.0.

Return type `None`

setPersistent(*key*)
Makes a key's value persistent through program restarts.

Parameters **key** (`str`) – the key to make persistent

New in version 2017.0.0.

Return type `None`

class networktables.**NetworkTableEntry**(*api*, *local_id*, *key*)
Holds a value from NetworkTables, and changes it as new entries come in. Do not create this object directly, use `NetworkTablesInstance.getEntry()` or *NetworkTable.getEntry()* to obtain an instance of this class.

Using NetworkTableEntry objects to access/change NT values is more efficient than the getX/putX methods of *NetworkTable*.

New in version 2018.0.0.

addListener(*listener*, *flags*, *paramIsNew=True*)
Add a listener for changes to the entry

Parameters

- **listener** (*callable(entry, key, value, param)*) – the listener to add
- **flags** (*NetworkTablesInstance.NotifyFlags*) – bitmask specifying desired notifications
- **paramIsNew** (bool) – If True, the listener fourth parameter is a boolean set to True if the listener is being called because of a new value in the table. Otherwise, the parameter is an integer of the raw *NT_NOTIFY_** flags

Returns listener handle

clearFlags(*flags*)
Clears flags

Parameters **flags** (int) – the flags to clear (bitmask)

Return type None

clearPersistent()
Stop making value persistent through program restarts.

Return type None

delete()
Deletes the entry.

Return type bool

exists()
Determines if the entry currently exists

Return type bool

forceSetBoolean(*value*)
Sets the entry's value.

Parameters **value** (bool) – the value to set

forceSetBooleanArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[bool]) – the value to set

forceSetDouble(*value*)
Sets the entry's value.

Parameters **value** (float) – the value to set

forceSetDoubleArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[float]) – the value to set

forceSetNumber(*value*)
Sets the entry's value.

Parameters **value** (float) – the value to set

forceSetNumberArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[float]) – the value to set

forceSetRaw(*value*)
Sets the entry's value.

Parameters **value** (bytes) – the value to set

forceSetString(*value*)
Sets the entry's value.

Parameters **value** (str) – the value to set

forceSetStringArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[str]) – the value to set

forceSetValue(*value*)
Sets the entry's value

Parameters **value** – the value that will be assigned

> **Warning:** Empty lists will fail

get()

getBoolean(*defaultValue*)
Gets the entry's value as a boolean. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[bool, ~D]

Returns the entry's value or the given default value

getBooleanArray(*defaultValue*)
Gets the entry's value as a boolean array. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[Sequence[bool], ~D]

Returns the entry's value or the given default value

getDouble(*defaultValue*)
Gets the entry's value as a double. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[float, ~D]

Returns the entry's value or the given default value

getDoubleArray(*defaultValue*)
Gets the entry's value as a double array. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[Sequence[float], ~D]

Returns the entry's value or the given default value

getFlags()
Returns the flags.

Return type int

Returns the flags (bitmask)

getHandle()
Gets the native handle for the entry

getInfo()
Gets combined information about the entry.

Returns Entry information

Return type tuple of (name, type, flags)

getName()
Gets the name of the entry (the key)

Return type str

getNumber(*defaultValue*)
Gets the entry's value as a double. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[float, ~D]

Returns the entry's value or the given default value

getRaw(*defaultValue*)
Gets the entry's value as a raw value (byte array). If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[bytes, ~D]

Returns the entry's value or the given default value

getString(*defaultValue*)
Gets the entry's value as a string. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Return type Union[str, ~D]

Returns the entry's value or the given default value

getStringArray(*defaultValue*)
Gets the entry's value as a string array. If the entry does not exist or is of different type, it will return the default value.

Parameters **defaultValue** (*~D*) – the value to be returned if no value is found

Returns the entry's value or the given default value

Return type list(float)

getType()
Gets the type of the entry

Return type *NetworkTablesInstance.EntryTypes*

isPersistent()
Returns whether the value is persistent through program restarts.

Return type bool

Returns True if the value is persistent.

classmethod isValidDataType(*data*)

key

removeListener(*listener_id*)
Remove a listener from receiving entry events

Parameters **listener** – the callable that was passed to addListener

Return type None

setBoolean(*value*)
Sets the entry's value.

Parameters **value** (bool) – the value to set

Return type bool

Returns False if the entry exists with a different type

setBooleanArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[bool]) – the value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultBoolean(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (bool) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultBooleanArray(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (Sequence[bool]) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultDouble(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (float) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultDoubleArray(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (Sequence[float]) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultNumber(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (float) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultNumberArray(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (Sequence[float]) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultRaw(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (bytes) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultString(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (str) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultStringArray(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** (Sequence[str]) – the default value to set

Return type bool

Returns False if the entry exists with a different type

setDefaultValue(*defaultValue*)
Sets the entry's value if it does not exist.

Parameters **defaultValue** – the default value to set

Return type bool

Returns False if the entry exists with a different type

> **Warning:** Do not set an empty list, it will fail

setDouble(*value*)
Sets the entry's value.

Parameters **value** (float) – the value to set

Return type bool

Returns False if the entry exists with a different type

setDoubleArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[float]) – the value to set

Return type bool

Returns False if the entry exists with a different type

setFlags(*flags*)
Sets flags.

Parameters **flags** (int) – the flags to set (bitmask)

Return type None

setNumber(*value*)
Sets the entry's value.

Parameters **value** (float) – the value to set

Return type bool

Returns False if the entry exists with a different type

setNumberArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[float]) – the value to set

Return type bool

Returns False if the entry exists with a different type

setPersistent()
Make value persistent through program restarts.

Return type None

setRaw(*value*)
Sets the entry's value.

Parameters **value** (bytes) – the value to set

Return type bool

Returns False if the entry exists with a different type

setString(*value*)
Sets the entry's value.

Parameters **value** (str) – the value to set

Return type bool

Returns False if the entry exists with a different type

setStringArray(*value*)
Sets the entry's value.

Parameters **value** (Sequence[str]) – the value to set

Return type bool

Returns False if the entry exists with a different type

setValue(*value*)
Sets the entry's value

Parameters **value** – the value that will be assigned

Return type bool

Returns False if the table key already exists with a different type

> **Warning:** Empty lists will fail

property value
Property to access the value of this entry, or None if the entry hasn't been initialized yet (use setXXX or forceXXX)

1.4 Utilities

networktables.util.**ntproperty**(*key*, *defaultValue*, *writeDefault=True*, *doc=None*, *persistent=False*, ***, *inst=<_pynetworktables.instance.NetworkTablesInstance object>*)

A property that you can add to your classes to access NetworkTables variables like a normal variable.

Parameters

- **key** (str) – A full NetworkTables key (eg /SmartDashboard/foo)
- **defaultValue** (*any*) – Default value to use if not in the table
- **writeDefault** (bool) – If True, put the default value to the table, overwriting existing values
- **doc** (Optional[str]) – If given, will be the docstring of the property.
- **persistent** (bool) – If True, persist set values across restarts. *writeDefault* is ignored if this is True.
- **inst** (NetworkTablesInstance) – The NetworkTables instance to use.

Example usage:

```
class Foo(object):

    something = ntproperty('/SmartDashboard/something', True)

    ...

    def do_thing(self):
        if self.something:    # reads from value
            ...

            self.something = False # writes value
```

Note: Does not work with empty lists/tuples.

Getting the value of this property should be reasonably fast, but setting the value will have just as much overhead as *NetworkTable.putValue()*

> **Warning:** This function assumes that the value's type never changes. If it does, you'll get really strange errors... so don't do that.

New in version 2015.3.0.

Changed in version 2017.0.6: The *doc* parameter.

Changed in version 2018.0.0: The *persistent* parameter.

Return type property

class networktables.util.**ChooserControl**(*key*, *on_choices=None*, *on_selected=None*, ***, *inst=<_pynetworktables.instance.NetworkTablesInstance object>*)

Interacts with a wpilib.SendableChooser object over NetworkTables.

Parameters

- **key** (str) – NetworkTables key
- **on_choices** (Optional[Callable[[Sequence[str]], None]]) – A function that will be called when the choices change.
- **on_selection** – A function that will be called when the selection changes.
- **inst** (NetworkTablesInstance) – The NetworkTables instance to use.

close()

Stops listening for changes to the SendableChooser

Return type None

getChoices()

Returns the current choices. If the chooser doesn't exist, this will return an empty tuple.

Return type Sequence[str]

getSelected()

Returns the current selection or None

Return type Optional[str]

setSelected(*selection*)

Sets the active selection on the chooser

Parameters **selection** (str) – Active selection name

Return type None

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

INDEX

G

I

K

L

N

P

R

S

T

U

V

W