RobotPy networktables Documentation

Release 2017.0.4

RobotPy Development Team

January 18, 2017

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 1

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)

sd = NetworkTables.getTable("SmartDashboard")

i = 0
while True:
    try:
        print('dsTime:', sd.getNumber('dsTime'))
    except KeyError:
        print('dsTime: N/A')

    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:
    try:
        print('robotTime:', sd.getNumber('robotTime'))
    except KeyError:
        print('robotTime: N/A')

    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

# 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.addTableListener(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)

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:
    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.addGlobalListener(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
```

```
# 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')
    print('robotTime:', c.robotTime)

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

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

1.2 NetworkTables API

class networktables.**NetworkTables**

This is the global singleton that you use to initialize NetworkTables connections, configure global settings and listeners, and to create NetworkTable instances 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')
...
```

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

BOOLEAN = '\x01'
True or False

BOOLEAN_ARRAY = '\x10'
List of booleans

DOUBLE = '\x02'
Floating point number

DOUBLE_ARRAY = ' '
List of numbers

RAW = '\x08'
Raw bytes

STRING = '\x04'
Strings

STRING_ARRAY = '@'
List of strings

class NetworkTables.**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

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

classmethod NetworkTables.**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

classmethod NetworkTables.**addGlobalListener**(*listener*, *immediateNotify=True*, *localNotify=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.

This will automatically initialize network tables if it has not been already.

Parameters

- **listener** – A callable that has this signature: *callable(key, value, isNew)*
- **immediateNotify** – If True, the listener will be called immediately with the current values of the table
- **immediateNotify** – bool

New in version 2015.2.0.

Changed in version 2017.0.0: *paramIsNew* parameter added

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

classmethod NetworkTables.**addGlobalListenerEx**(*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.

This will automatically initialize network tables if it has not been already.

Parameters

- **listener** – 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

classmethod NetworkTables.**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.

classmethod NetworkTables.**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.

classmethod NetworkTables.**getGlobalAutoUpdateValue**(*key*, *defaultValue*, *writeDefault*)
Global version of getAutoUpdateValue. This function will not initialize NetworkTables.

Parameters

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

New in version 2015.3.0.

See also:

ntproperty() is a read-write alternative to this

classmethod NetworkTables.**getGlobalTable**()
Returns an object that allows you to write values to raw network table keys (which are paths with / separators).

This will automatically initialize network tables if it has not been already.

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

Return type *NetworkTable*

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

Returns IP address of server or None

Return type str

New in version 2015.3.2.

classmethod NetworkTables.**getTable**(*key*)
Gets the table with the specified key. If the table does not exist, a new table will be created.

This will automatically initialize network tables if it has not been already initialized.

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

Returns the network table requested

Return type *NetworkTable*

classmethod NetworkTables.**globalDeleteAll**()
Deletes ALL keys in ALL subtables.

Warning: Use with caution!

New in version 2017.0.0.

classmethod NetworkTables.**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:

```
cls.setIPAddress(server)
cls.setClientMode()
cls.initialize()
```

New in version 2017.0.0: The *server* parameter

NetworkTables.**ipAddress = None**

classmethod NetworkTables.**isConnected**()

Returns True if connected to at least one other NetworkTables instance

Return type bool

classmethod NetworkTables.**isServer**()

Returns True if configured in server mode

classmethod NetworkTables.**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.

NetworkTables.**port = 1735**

classmethod NetworkTables.**removeConnectionListener**(*listener*)
Removes a connection listener

Parameters **listener** – The function registered for connection notifications

classmethod NetworkTables.**removeGlobalListener**(*listener*)
Removes a global listener

New in version 2015.2.0.

classmethod NetworkTables.**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.

classmethod NetworkTables.**setClientMode**()
Set that network tables should be a client

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**setDashboardMode**()
This will allow the driver station to connect to your code and receive the IP address of the robot from it. You must not call setClientMode(), setTeam(), or setIPAddress()

Warning: Only use this if your pynetworktables client is running on the same host as the driver station, or nothing will happen!
This mode will only connect to the robot if the FRC Driver Station is able to connect to the robot and the LabVIEW dashboard has been disabled.

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**setIPAddress**(*address*)

Parameters **address** (*str*) – the adress that network tables will connect to in client mode

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**setNetworkIdentity**(*name*)
Sets the network identity. This 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.

classmethod NetworkTables.**setPersistentFilename**(*filename*)
Sets the persistent filename. Not used on the client.

Parameters **filename** (*str*) – the filename that the network tables server uses for automatic loading and saving of persistent values

New in version 2017.0.0.

classmethod NetworkTables.**setPort**(*port*)
Sets the port number that network tables will connect to in client mode or listen to in server mode.

Parameters **port** (*int*) – the port number

New in version 2017.0.0.

classmethod NetworkTables.**setServerMode**()
set that network tables should be a server (this is the default)

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**setTeam**(*team*)
set the team the robot is configured for (this will set the ip address that network tables will connect to in client mode)

Parameters **team** (*str*) – the team number

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**setTestMode**(*server=True*)
Setup network tables to run in unit test mode, and enables verbose logging.

Warning: This must be called before initalize() or getTable()

classmethod NetworkTables.**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.

classmethod NetworkTables.**setWriteFlushPeriod**(*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.

classmethod NetworkTables.**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.

1.3 NetworkTable Instances

class networktables.**NetworkTable**(*path*, *api*)
This is a NetworkTable instance, 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')

sd.putNumber('someNumber', 1234)
...
```

See also:

- The examples in the documentation.
- *NetworkTables*

PATH_SEPARATOR = '/'

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 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

addTableListener(*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** – 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** (*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)

addTableListenerEx(*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** – 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** (*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.

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.

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

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

New in version 2017.0.0.

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

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

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

Return type bool

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)

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

Return type bool

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

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

New in version 2017.0.0.

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 *AutoUpdateValue*

Note: If you modify the returned value, the value will NOT be written back to NetworkTables. 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.

getBoolean(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)

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 the key is None. If not specified, raises KeyError if the key is None.

Returns the key

Return type bool

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

getBooleanArray(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)

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** (*list(bool)*) – the value to be returned if no value is found

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

Return type list(bool)

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

New in version 2017.0.0.

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".

Returns keys currently in the table

Return type list

New in version 2017.0.0.

getNumber(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)

Gets the number associated with the given name.

Parameters

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

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

Return type int, float

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

getNumberArray(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)

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** (*list(int, float)*) – the value to be returned if no value is found

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

Return type list(int, float)

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

New in version 2017.0.0.

getRaw(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)

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

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

Return type bytes

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

New in version 2017.0.0.

getString(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)
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

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

Return type str

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

getStringArray(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)
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** (*list(str)*) – the value to be returned if no value is found

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

Return type list(str)

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

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

Returns the networktable to be returned

Return type *NetworkTable*

getSubTables()

Returns subtables currently in the table

Return type list

New in version 2017.0.0.

getValue(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)
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, list

Raises **KeyError** – If the value doesn't exist and no default is provided, or if it is the wrong type

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.

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

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

Return type bool

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

Parameters

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

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

Return type bool

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** (*int, float*) – the value that will be assigned

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

Return type bool

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

Parameters

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

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

Return type bool

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

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

Return type bool

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

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

Return type bool

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

Parameters

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

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

Return type bool

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*	Doesn't work in Python 2.7
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

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

Return type bool

New in version 2017.0.0.

removeTableListener(*listener*)
Removes a table listener

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

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.

Returns False if the table key exists with a different type

New in version 2017.0.0.

setDefaultBooleanArray(*key*, *defaultValue=<class networktables.networktable._defaultValueSentry>*)
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** (*list(bool)*) – the default value to set if key doesn't exist.

Returns False if the table key exists with a different type

Return type bool

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.

Returns False if the table key exists with a different type

Return type bool

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** (*list(int, float)*) – the default value to set if key doesn't exist.

Returns False if the table key exists with a different type

Return type bool

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.

Returns False if the table key exists with a different type

Return type bool

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.

Returns False if the table key exists with a different type

Return type bool

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** (*list(str)*) – the default value to set if key doesn't exist.

Returns False if the table key exists with a different type

Return type bool

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.

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.

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.

class networktables.autovalue.**AutoUpdateValue**(*key*, *default*, *valuefn*)
Holds a value from NetworkTables, and changes it as new entries come in. Updates to this value are NOT passed on to NetworkTables.

Do not create this object directly, as it only holds the value. Use `NetworkTables.getAutoUpdateValue()` to obtain an instance of this.

get()
Returns the value held by this object

value

1.4 Utilities

networktables.util.**ntproperty**(*key*, *defaultValue*, *writeDefault=True*)
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

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: When using python 2.x, the property must be assigned to a new-style class or it won't work!

New in version 2015.3.0.

class networktables.util.**ChooserControl**(*key*, *on_choices=None*, *on_selected=None*)
Interacts with a `wpilib.sendablechooser.SendableChooser` object over NetworkTables.

Parameters

- **key** (*str*) – NetworkTables key
- **on_choices** – A function that will be called when the choices change. Signature: fn(value)
- **on_selection** – A function that will be called when the selection changes. Signature: fn(value)

close()
Stops listening for changes to the SendableChooser

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

Return type tuple

getSelected()
Returns the current selection or None

Return type str

setSelected(*selection*)
Sets the active selection on the chooser

Parameters **selection** – Active selection name

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

n

I

L

N

P

R

S

U

V