Robot networking: using NetworkTables

Last Updated: 09-11-2018

Table of Contents

Getting started with NetworkTables

What is NetworkTables

NetworkTables is an implementation of a distributed "dictionary". That is named values are created either on the robot, driver station, or potentially an attached coprocessor, and the values are automatically distributed to all the other participants. For example, a driver station laptop might receive camera images over the network, perform some vision processing algorithm, and come up with some values to sent back to the robot. The values might be an X, Y, and Distance. By writing these results to NetworkTable values called "X", "Y", and "Distance" they can be read by the robot shortly after being written. Then the robot can act upon them.

NetworkTables can be used by programs on the robot in either C++, Java or LabVIEW and is built into each version of WPILib.

Network tables organization

Data is organized in NetworkTables in a hierarchy much like a directory on disk with folders and files. For any instance of NetworkTables there can be multiple values and subtables that may be nested in whatever way fits the data organization desired. Subtables actually are represented as a long key with slashes (/) separating the nested subtable and value key names. Each value has a key associated with it that is used to retrieve the value. For example, you might have a table called **datatable** as shown in the following examples. Within **datatable** there are two keys, X and Y and their associated values. The OutlineViewer is a good utility for exploring the values stored in NetworkTables while a program is running.

Data types for NetworkTables are either boolean, numeric, or string. Numeric values are written as double precision values. In addition you can have arrays of any of those types to ensure that multiple data items are delivered consistently. There is also the option of storing raw data which can be used for representing structured data.

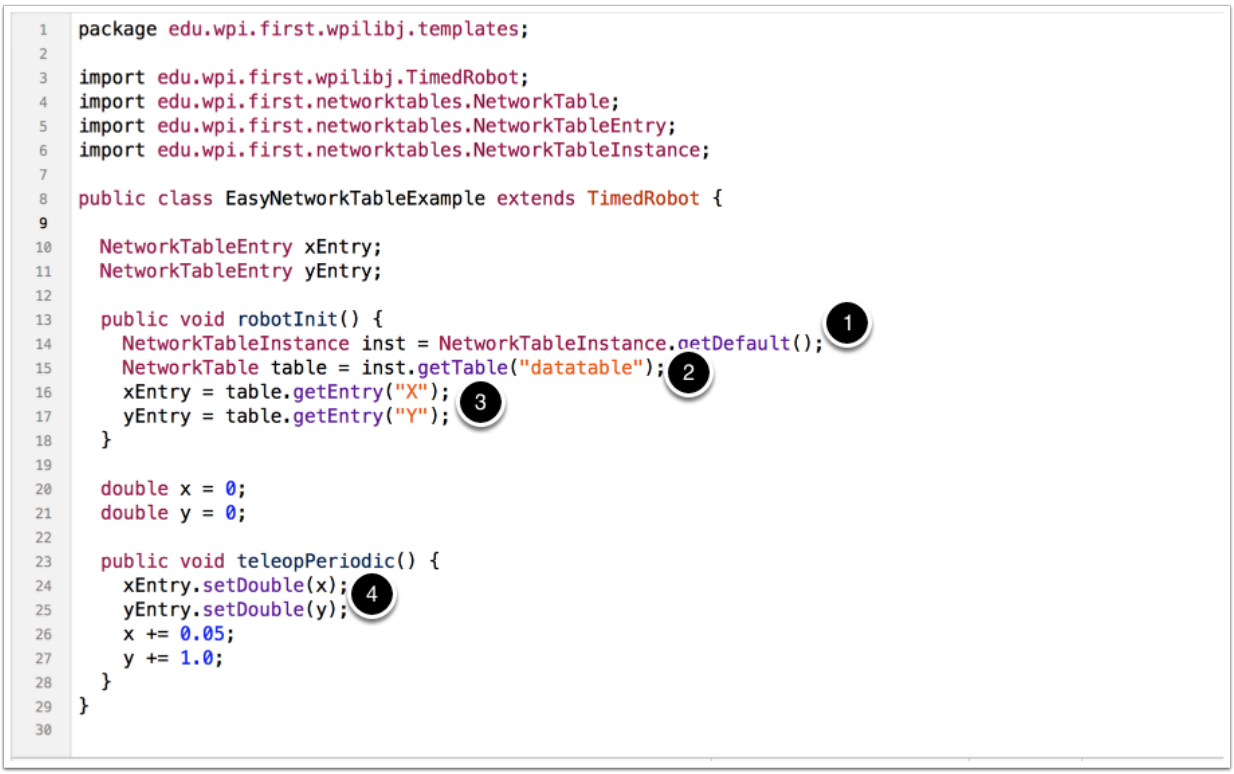
There are some default tables that are created automatically when the program starts up:

Robot networking: using NetworkTables

Table name	Use
/SmartDashboard	Used to store values written to the SmartDashboard or Shuffleboard using the SmartDashboard.put() set of methods.
/LiveWindow	Used to store Test mode (Test on the Driver Station) values. Typically these are Subsystems and the associated sensors and actuators.
/FMSInfo	Information about the currently running match that comes from the Driver Station and the Field Management System.

Writing a simple NetworkTable program

The NetworkTables classes are instantiated automatically when your program starts. To access the instance of NetworkTables do call methods to read and write the getDefault() method can be used to get the default instance.



```
1  package edu.wpi.first.wpilibj.templates;
2
3  import edu.wpi.first.wpilibj.TimedRobot;
4  import edu.wpi.first.networktables.NetworkTable;
5  import edu.wpi.first.networktables.NetworkTableEntry;
6  import edu.wpi.first.networktables.NetworkTableInstance;
7
8  public class EasyNetworkTableExample extends TimedRobot {
9
10   NetworkTableEntry xEntry;
11   NetworkTableEntry yEntry;
12
13   public void robotInit() {
14     NetworkTableInstance inst = NetworkTableInstance.getDefault();   (1)
15     NetworkTable table = inst.getTable("datatable");                 (2)
16     xEntry = table.getEntry("X");                                    (3)
17     yEntry = table.getEntry("Y");
18   }
19
20   double x = 0;
21   double y = 0;
22
23   public void teleopPeriodic() {
24     xEntry.setDouble(x);                                             (4)
25     yEntry.setDouble(y);
26     x += 0.05;
27     y += 1.0;
28   }
29 }
30
```

1. Get the default instance of NetworkTables that was created automatically when your program starts.

2. Get the table within that instance that contains the data. There can be as many tables as you like and exist to make it easier to organize your data. In this case it's a table called dataTable.
3. Get the entries within that table that correspond to the X and Y values for some operation in your program.
4. Using the entry objects, set the value to a double that is constantly increasing. The keys are actually "/datatable/X" and "/datatable/Y". If they don't already exist, the key/value pair is added.

The values for X and Y can be easily viewed using the OutlineViewer program that shows the NetworkTables hierarchy and all the values associated with each key.

Actually network tables has a flat namespace for the keys. Having tables and subtables is an abstraction to make it easier to organize your data. So for a table called "SmartDashboard" and a key named "xValue", it is really a single key called "/SmartDashboard/xValue". The hierarchy is not actually represented in the distributed data, only keys with prefixes that are the contained table.

C++ version of the program

```
#include "TimedRobot.h"
#include "networktables/NetworkTable.h"
#include "networktables/NetworkTableEntry.h"
#include "networktables/NetworkTableInstance.h"

class EasyNetworkTableExample : public frc::TimedRobot {
 public:
  nt::NetworkTableEntry xEntry;
  nt::NetworkTableEntry yEntry;

  void RobotInit() {
    auto inst = nt::NetworkTableInstance::GetDefault();
    auto table = inst.GetTable("datatable");
    xEntry = table->GetEntry("X");
    yEntry = table->GetEntry("Y");
  }

  double x = 0;
  double y = 0;

  void TeleopPeriodic() {
    xEntry.SetDouble(x);
    yEntry.SetDouble(y);
    x += 0.05;
    y += 1.0;
  }
};

START_ROBOT_CLASS(EasyNetworkTableExample)

```

Listening for value changes

A common use case for NetworkTables is where a program on the Driver Station generates values that need to be sent to the robot. For example, imagine that some image processing code running on the Driver Station computes the heading and distance to a goal and sends those values to the robot. In this case it might be desirable for the robot program to be notified when new values arrive from the Driver Station. NetworkTables provides facilities to do that.

Using a NetworkTable EntryListener

One of the more common cases for NetworkTables is waiting for a value to change on the robot. This is called an EntryListener because it notifies the robot program when a network table entry (single value) changes. The following code shows how to do this.

```
1  package networktablesdesktopclient;
2
3  import edu.wpi.first.networktables.EntryListenerFlags;
4  import edu.wpi.first.networktables.NetworkTable;
5  import edu.wpi.first.networktables.NetworkTableEntry;
6  import edu.wpi.first.networktables.NetworkTableInstance;
7
8  public class TableEntryListenerExample {
9
10   public static void main(String[] args) {
11     new TableEntryListenerExample().run();
12   }
13
14   public void run() {
15     NetworkTableInstance inst = NetworkTableInstance.getDefault();
16     NetworkTable table = inst.getTable("datatable");
17     NetworkTableEntry yEntry = table.getEntry("Y");
18     inst.startClientTeam(190);
19
20     table.addEntryListener("X", (table, key, entry, value, flags) -> {
21       System.out.println("X changed value: " + value.getValue());
22     }, EntryListenerFlags.kNew | EntryListenerFlags.kUpdate);
23
24     yEntry.addListener(event -> {
25       System.out.println("Y changed value: " + event.value.getValue());
26     }, EntryListenerFlags.kNew | EntryListenerFlags.kUpdate);
27
28     try {
29       Thread.sleep(100000);
30     } catch (InterruptedException ex) {
31       System.out.println("interrupted!");
32       return;
33     }
34   }
35 }
36
```

1. Get the default instance of NetworkTables.
2. Get an reference to the subtable containing the keys that are to be watched for changes.
3. Get a reference to the Entry object the the particular key ("Y") that should be watched.

4. Add an EntryListener for changed values of "X". The lambda ("->" operator) defines the code that should run when "X" changes.
5. Add an EntryListener for changed values of "Y". The lambda defines the code that should run when the value of "Y" changes.

Creating a client-side program

If all you need to do is have your robot program communicate with GRIP or a dashboard running on the Driver Station laptop, then the previous examples of writing robot programs are sufficient. But if you would like to write some custom client code that would run on the drivers station or on a coprocessor then you need to know how to build Network Tables programs for those (non-roboRIO) platforms.

A basic client program looks like the following example.

```
package networktablesdesktopclient;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableEntry;
import edu.wpi.first.networktables.NetworkTableInstance;

public class NetworkTablesDesktopClient {
  public static void main(String[] args) {
    new NetworkTablesDesktopClient().run();
  }

  public void run() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();
    NetworkTable table = inst.getTable("datatable");
    NetworkTableEntry xEntry = table.getEntry("x");
    NetworkTableEntry yEntry = table.getEntry("y");
    inst.startClientTeam(TEAM);  // where TEAM=190, 294, etc, or use inst.
startClient("hostname") or similar
    inst.startDSClient();  // recommended if running on DS computer; this gets the robot
IP from the DS
    while (true) {
      try {
        Thread.sleep(1000);
      } catch (InterruptedException ex) {
        System.out.println("interrupted");
        return;
      }
      double x = xEntry.getDouble(0.0);
      double y = yEntry.getDouble(0.0);
      System.out.println("X: " + x + " Y: " + y);
```

```
      }
    }
}
```

In this example an instance of NetworkTables is created and a NetworkTableEntry if created to reference the values of "x" and "y" from a table called "datatable".

Then this instance is started as a NetworkTable client with the team number (the roboRIO is always the server). Additionally, if the program is running on the Driver Station computer, by using the startDSClient() method, NetworkTables will get the robot IP address from the Driver Station.

Then this sample program simply loops once a second and gets the values for x and y and prints them on the console. In a more realistic program, the client might be processing or generating values for the robot to consume.

Building the program

When building and running the program you will need some additional libraries to include with your client-side program. These are:

http://first.wpi.edu/FRC/roborio/mav...java-4.0.0.jar (ntcore java files)

http://first.wpi.edu/FRC/roborio/mav...-4.0.0-all.jar (ntcore native libs for all desktop platforms)

http://first.wpi.edu/FRC/roborio/mav...java-3.0.0.jar (wpiutil java files)

Note: The desktop platform jar is for Windows, Mac, and Linux.

Advanced uses of NetworkTables

Creating multiple instances of NetworkTables

This feature is mainly useful for coprocessors and unit testing. It allows a single program to be a member of two completely independent Networktables "networks" that contain completely different (and unrelated) sets of tables. For most general usage, you should use tables within the single instance, as all current dashboard programs can only connect to a single NT server at a time.

Normally the "default" instance is set up on the robot as a server, and used for communication with the dashboard program running on your driver station computer. This is what the SmartDashboard and LiveWindow classes use.

If you had a coprocessor and wanted to have a set of tables that's shared only between the coprocessor and the robot, you could set up a separate instance in the robot code that acts as a client (or a server) and connect the coprocessor to it, and those tables will NOT be sent to the dashboard.

Similarly, if you wanted to do unit testing of your robot program's NT communications, you could set up your unit tests such that they create a separate client instance (still within the same program) and have it connect to the server instance that the main robot code is running.

Another example might be having two completely separate dashboard programs. You could set up two networktables server instances in your robot program (on different TCP ports of course), set up different tables on each one, and have each dashboard connect to a different server instance. Each dashboard would only see the tables on its instance.

Listening to changes on specific keys

```java
package networktablesdesktopclient;

import edu.wpi.first.networktables.EntryListenerFlags;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableEntry;
import edu.wpi.first.networktables.NetworkTableInstance;

public class TableEntryListenerExample {

  public static void main(String[] args) {
    new TableEntryListenerExample().run();
  }

  public void run() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();
    NetworkTable table = inst.getTable("datatable");
    NetworkTableEntry yEntry = table.getEntry("Y"); (1)
    inst.startClientTeam(190);

    table.addEntryListener("X", (table, key, entry, value, flags) -> {
      System.out.println("X changed value: " + value.getValue());
    }, EntryListenerFlags.kNew | EntryListenerFlags.kUpdate); (2)

    yEntry.addListener(event -> {
      System.out.println("Y changed value: " + event.value.getValue());
    }, EntryListenerFlags.kNew | EntryListenerFlags.kUpdate); (3)

    try {
      Thread.sleep(100000);
    } catch (InterruptedException ex) {
      System.out.println("interrupted!");
      return;
    }
  }
}
```

This program shows how to listen for changes on two keys in NetworkTables using two different methods. One listens for changes of the key "X" in "/datatable" and the other listens for changes to the key "Y" in "datatable".

1. Create a NetworkTableEntry object that refers to "/datatable/Y".
2. Using a lambda, print a message every time the value "/datatable/X" is created or updated.
3. Again using a lambda, print a message every time the value of "/datatable/Y" is created or changed.

Listening to changes in a table

```java
package networktablesdesktopclient;

import edu.wpi.first.networktables.EntryListenerFlags;
import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableInstance;

public class TableEntryListenerExample {

  public static void main(String[] args) {
    new TableEntryListenerExample().run();
  }

  public void run() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();
    NetworkTable table = inst.getTable("datatable"); (1)
    inst.startClientTeam(190);

    table.addEntryListener((table, key, entry, value, flags) -> { (2)
      System.out.println("Key: " + key + " Value: " + value.getValue() + " Flags: " + flags);
    }, EntryListenerFlags.kNew | EntryListenerFlags.kUpdate);

    try {
      Thread.sleep(100000);
    } catch (InterruptedException ex) {
      System.out.println("interrupted!");
      return;
    }
  }
}
```

This program is similar to the previous one, but it is looking for any changes to the table "/datatable". When an update or a new value is added to the table the key and value are printed.

1. Create a NetworkTable object for "/datatable"
2. Listen for additions and updates to keys in the table and print the key and value of the new entry.

Listening for subtable creation

```java
package networktablesdesktopclient;

import edu.wpi.first.networktables.NetworkTable;
import edu.wpi.first.networktables.NetworkTableInstance;

public class SubTableListenerExample {

  public static void main(String[] args) {
    new SubTableListenerExample().run();
  }

  public void run() {
    NetworkTableInstance inst = NetworkTableInstance.getDefault();
    NetworkTable table = inst.getTable("datatable");
    inst.startClientTeam(190);

    table.addSubTableListener((parent, name, table) -> {
      System.out.println("Parent: " + parent + " Name: " + name);
    }, false);

    try {
      Thread.sleep(100000);
    } catch (InterruptedException ex) {
      System.out.println("interrupted!");
      return;
    }
  }
}
```

In this example a listener is defined that listens for creation of subtables one level down from the listener. That is subtables with the key "/datatable/<any-table-name>". This will fire the callback for each existing subtable, then continue to listen for new subtables created immediately below "datatable" in this case.