**EECE 4263: Embedded Systems** **Name: ______________________**

# Lab Assignment 5 — *"I've seen the [LED] light!"* Introduction to MQTT, JSON and MQTT-Spy

## Overview

In this lab you'll use the Message Queuing Telemetry Transport (**MQTT**) protocol to communicate wirelessly with a remote "LED node," sending messages to turn the node's LED on or off and receiving messages from the LED node confirming the LED's current state.

Developed by IBM, MQTT is a small-footprint publisher-subscriber ("pub-sub") messaging protocol used for M2M (*machine-to-machine*) communication where memory and bandwidth are limited. MQTT is implemented as a middleware protocol using the TCP and IP layers of the Internet stack to send and receive data payloads via TCP port 1883.

Although payload format is not part of the MQTT specification (MQTT simply transports groups of characters), payload data is often formatted using JavaScript Object Notation (**JSON**). JSON is a powerful, lightweight, and simple data-interchange format easily read by both humans and machines, making it an ideal payload format for MQTT. You'll be using JSON in this lab.

Additionally, this lab will give you experience using the open-source **MQTT-Spy** application to send, receive, filter, and monitor MQTT messages (i.e., it's an MQTT "message sniffer").

Finally, you'll interact with a Linux-based MQTT broker (running on a remote Raspberry-Pi), becoming familiar with the MQTT architecture, components, and messaging formats. This knowledge will be invaluable as you work on future Internet-of-Things (IoT) projects.

## Prelab

Use the links below to familiarize yourself with the various communications protocols, formats, and applications needed to complete this lab. Start with the "**MQTT Essentials**" link, reading the "**Introducing MQTT**", "**Publish & Subscribe Basics**", and "**Topics & Best Practices**" sections. These will give you most of the background needed to understand the MQTT architecture. Next, look at "**The JSON standard**" link, where you'll learn how to create properly formatted JSON data structures. Finally, since you'll be using MQTT-Spy, look at the "**MQTT-Spy**" link as well.

MQTT
- MQTT Essentials: www.hivemq.com/mqtt-essentials/
- The MQTT standard: www.mqtt.org
- The MQTT community wiki: https://github.com/mqtt/mqtt.github.io/wiki

MQTT-Spy
- The MQTT-Spy info page: https://github.com/eclipse/paho.mqtt-spy
- MQTT-Spy download page: https://github.com/eclipse/paho.mqtt-spy/releases

JSON
- The JSON standard: www.json.org
- JSON example code: www.json.org/example

*NB:* **ArduinoJson** library 6.x.x and above is incompatible with the MQTT library. Recommended version is **5.13.3**.

After becoming familiar with MQTT, MQTT-Spy, and JSON, answer the questions below.

1. MQTT is a publisher-subscriber messaging transport protocol. Given several MQTT clients that wish to communicate on a variety of topics, explain the role of the **MQTT broker**.

2. Various Lipscomb administrative offices want to send and receive MQTT messages regarding students. Create a **hierarchical MQTT topic structure** that allows messages to be filtered by student's class standings (freshman, sophomores, etc.), colleges, and majors. **Don't worry about the payloads** for now—just develop a valid MQTT **topic** structure.

3. Assuming the messages in question 2. will contain a student's L-number and current grade point average, give two example **payloads** in proper JSON format.

4. A certain MQTT-based system exchanges messages tagged with the following topics: **animals/dogs/poodle**, **animals/dogs/collie**, **animals/horses/mustang**, **cars/nissan/gtr**, and **cars/ford/mustang**. Using proper MQTT wildcards, answer these questions:

   a. What single topic would let you only receive messages about dogs (all kinds)?

   b. What single topic would let you only receive messages about mustangs (horses or cars)?

   c. What single topic would let you receive *all* messages?

## Lab Procedure

Checkout the correct GitHub Classroom repo to download the files you will need to start this lab.

### LED Node

The goal of this lab is to use MQTT to communicate between two NodeMCU-based systems. At the heart of any MQTT system is the *broker*, and the broker requires each publisher/subscriber to have a unique Client ID. Throughout this lab, when you see a *ClientID* that ends with **XX**, replace **XX with the two-digit ID number of your kit!**

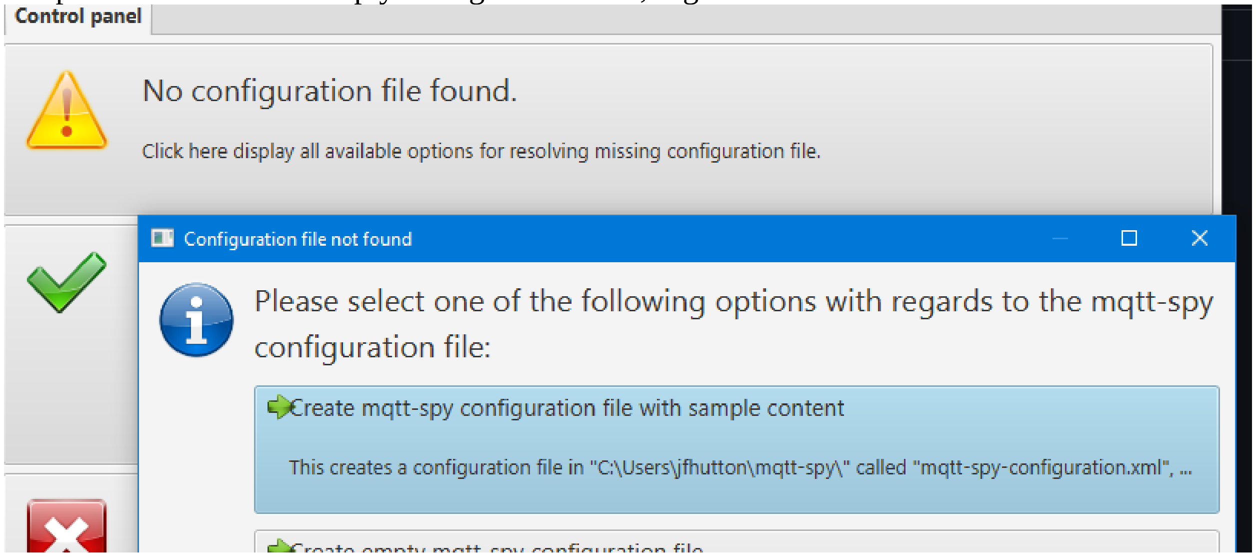
(e.g., with kit 03, **ledNodeXX** becomes **ledNode03**).

Construct an "**LED node**" by connecting an LED in series with a 330Ω current-limiting resistor between pin **D1** of your NodeMCU and ground. [*Active high? Active low? How-do-you-know?*] Your Lab05 GitHub repo contains a complete MQTT-based LED node program. Modify it with **your ClientID number**, compile it, and download it into the NodeMCU. Monitor its serial output using PuTTY (115200N81) to make sure it connects to the Lipscomb network, connects to the MQTT broker, and registers for topics with the broker. **Don't continue until this works**.

### MQTT-Spy

Next, open **MQTT-Spy**. MQTT-Spy is installed on the lab computers and is free to install on your own device as well. The latest *stable* version is **mqtt-spy-1.0.0.jar** (do **NOT** use any of the "beta" versions!) and can be downloaded at https://github.com/eclipse/paho.mqtt-spy/releases. There is also a version in your Initial Repo that should work. MQTT-Spy is a **Java app**, so you'll need to have Java installed as well.

Once MQTT-Spy is up and running, you may need to build a configuration file. Select the "with sample content" or the "empty configuration file", Figure 1.


Control panel
No configuration file found.
Click here display all available options for resolving missing configuration file.
Configuration file not found

Please select one of the following options with regards to the mqtt-spy configuration file:

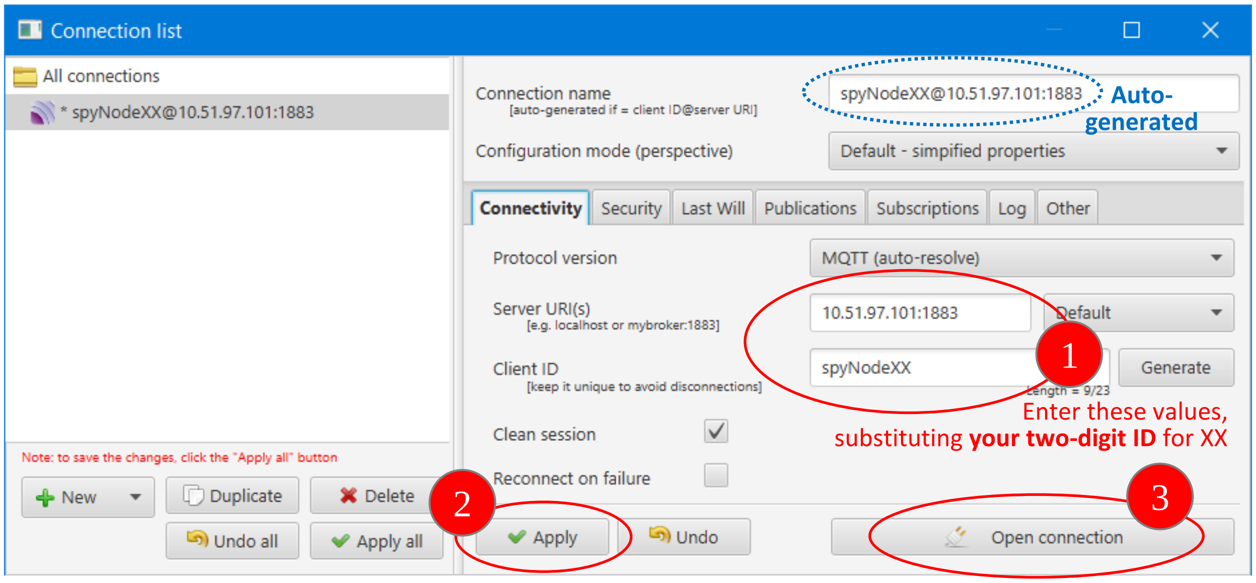
Create mqtt-spy configuration file with sample content

This creates a configuration file in "C:\Users\jfhutton\mqtt-spy\" called "mqtt-spy-configuration.xml", ...

Create empty mqtt-spy configuration file

Figure 1. Connecting to the ECE MQTT Broker

Now, use MQTT-Spy to connect to the ECE MQTT broker by selecting `Connections | New connection` from MQTT-Spy's top menu. A "Connection list" dialog box like the one shown in Figure 2 will appear.



Connection name [auto-generated if = client ID@server URI]: spyNodeXX@10.51.97.101:1883
Configuration mode (perspective): Default - simplified properties
Protocol version: MQTT (auto-resolve)
Server URI(s) [e.g. localhost or mybroker:1883]: 10.51.97.101:1883 Default
Client ID [keep it unique to avoid disconnections]: spyNodeXX
Clean session: ✓
Reconnect on failure: ☐

Figure 2. Connecting to the ECE MQTT Broker

Referring to Figure 2, set the Server URI(s) address to **10.51.97.101:1883** and the *ClientID* field to **spyNodeXX**, replacing **XX** with your two-digit ID number. Don't fill in the Connection name field—MQTT-Spy generates that automatically. Click "**Apply**" then "**Open connection**." The connection will appear as a new tab on the main MQTT-Spy panel. The panel will be YELLOW while MQTT-Spy attempts to connect to the broker, and GREEN once it's connected (e.g. Figure 3 below). If the tab turns RED it means MQTT-Spy was unable to connect to the broker. Should this happen, open the **Connections** menu, select **Manage connections**, and double-check the values you previously entered and attempt to reconnect. **Don't continue until this works**.

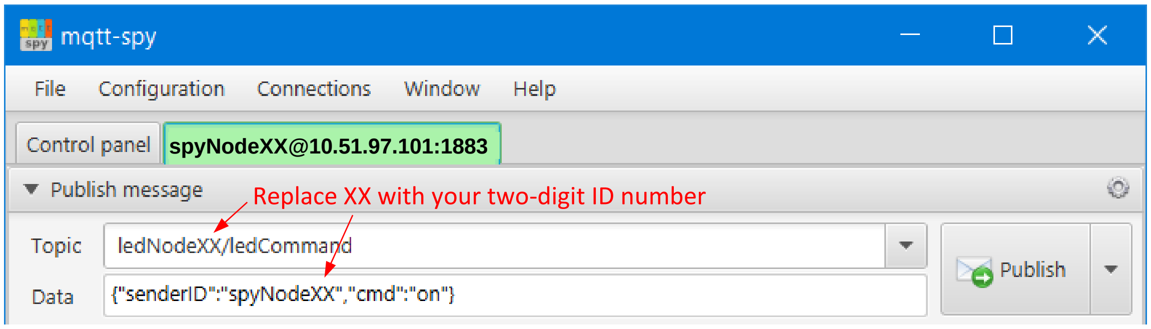
## MQTT LED On Command

The **ledNodeXX** node's LED can be turned on or off by sending the node an MQTT message with the topic **ledNodeXX/ledCommand** and payload data containing the sending node's ID and the command itself, both as JSON name-value pairs. Such a message can come from any node registered with the broker. MQTT-Spy is such a node, so let's use it to control the LED.

Referring to Figure 2, enter the **Topic** and **Data** values exactly as shown (using your ID number of course!). JSON is a language, so punctation is critical to syntax. Any code editor you have should support JSON which will do some of the checking for you. JSON also removes whitespace, so you can add to make more readable but still reduce to one line for MQTT-Spy.

Here is an example.

```
{
    "senderID":"spyNodeXX",
    "cmd":"on"
}
```


mqtt-spy

File Configuration Connections Window Help

Control panel | spyNodeXX@10.51.97.101:1883

Publish message

Replace XX with your two-digit ID number

Topic: ledNodeXX/ledCommand

Data: {"senderID":"spyNodeXX","cmd":"on"}

Publish

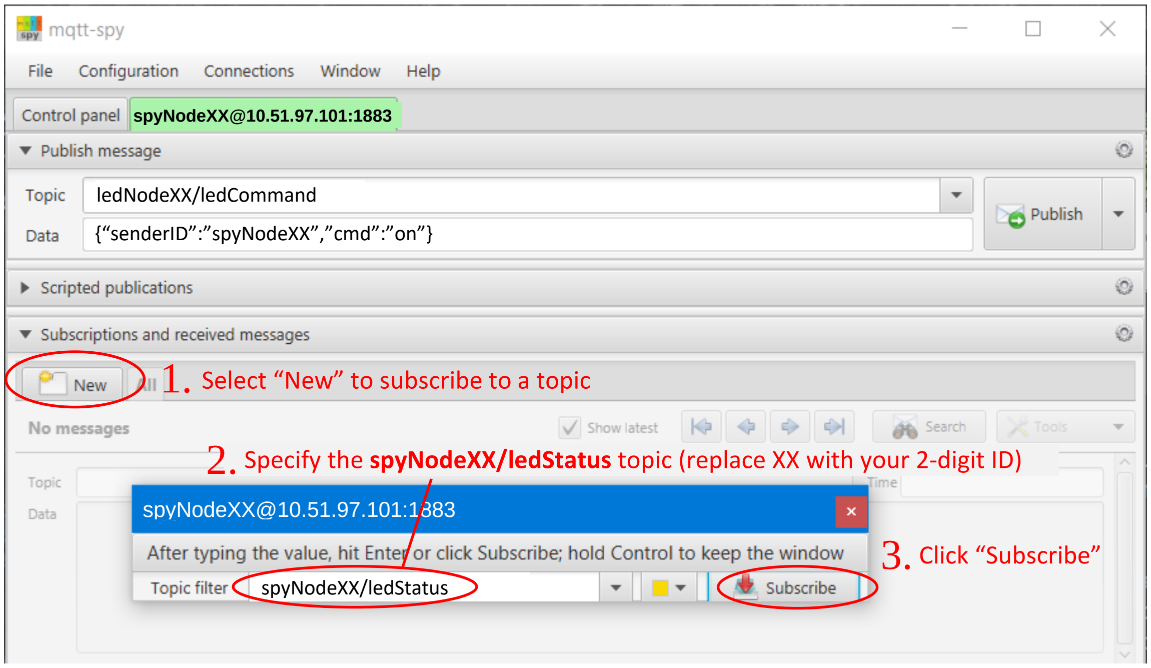
Figure 2. Sending a message to turn on the LED.

Once you click **Publish** to send the message **the LED should come on**. If not, double-check your wiring and the **Topic** and **Data** fields. **<u>Don't continue until this works</u>**.

Next examine the **mqtt_ledNode.ino** Arduino code to see what **cmd** value turns the LED off. Experiment with MQTT-Spy, turning the LED on and off, then ask yourself *"I wonder how hard would it be to control someone else's LED node?"* Hmmm.….

## MQTT Monitoring/Filtering

In addition to receiving messages, **ledNodeXX** sends messages as well. Whenever **ledNodeXX** receives a message to change the LED's state, **ledNodeXX** sends an **ledStatus** message back to the sending node. Figure 3 shows how to use MQTT Spy to subscribe to these status messages.


mqtt-spy

File Configuration Connections Window Help

Control panel | spyNodeXX@10.51.97.101:1883

Publish message

Topic: ledNodeXX/ledCommand

Data: {"senderID":"spyNodeXX","cmd":"on"}

Publish

Scripted publications

Subscriptions and received messages

New

1. Select "New" to subscribe to a topic

No messages

2. Specify the **spyNodeXX/ledStatus** topic (replace XX with your 2-digit ID)

spyNodeXX@10.51.97.101:1883

After typing the value, hit Enter or click Subscribe; hold Control to keep the window

Topic filter: spyNodeXX/ledStatus

Subscribe

3. Click "Subscribe"

Figure 3. Subscribing to receive LED status messages.

Once you've subscribed to **spyNodeXX/ledStatus** messages, a new tab with that name will appear to the right of the **All** tab. Click this new tab, then go back to the top of MQTT-Spy and send some "on" and "off" messages to **ledNodeXX**. As you do, you should see status messages received from **ledNodeXX** appear in the bottom window.

But how does **ledNodeXX** know which node to send the status message to? As seen above, the sender's name (**spyNodeXX** in this case) is part of the command message's payload, so all the **ledNodeXX** has to do is extract that name and construct a reply. But imagine if **ledNodeXX** were programmed more generally and simply sent **ledNodeXX/ledStatus** messages whenever the LED state changed. In that case, ANY node could register for **ledNodeXX/ledStatus** messages and would be notified when **ledNodeXX**'s LED changed state. This approach might be appropriate if the state of **ledNodeXX**'s LED were important to an entire group of nodes. But for this lab, the program designer determined that only the node that sent the command should receive the status message. This shows how configurable and adaptable MQTT messaging systems can be.

## MQTT Button Node

Complete this lab by constructing an MQTT-based **btnNodeXX** node using your second NodeMCU. The **btnNodeXX** node must have two pushbuttons ("On" and "Off"). Pressing either button should send a properly formatted MQTT **ledCommand** message to your **ledNodeXX** node, causing its LED to turn on or off. Also, your **btnNodeXX** node must receive **ledStatus** messages sent by the **ledNodeXX** node and display the message text on a serial monitor (115200N81) connected to **btnNodeXX**. Specs for the **ledNodeXX/ledCommand** and **btnNodeXX/ledStatus** messages are given below. Also, you will find the JSON syntax diagrams in Appendix A helpful for creating syntactically legal JSON objects.

```
  Topic: ledNodeXX/ledCommand
  Usage: Instructs ledNodeXX to turn on/off its LED
Payload: {"senderID":"btnNodeXX","cmd":"on"|"off"}

  Topic: btnNodeXX/ledStatus
  Usage: Current status of ledNodeXX node's LED
Payload: {"ledStatus":"on"|"off", "msg":"some message text"}
```

Study the LED node program to understand how MQTT messages are sent and received programmatically. Understanding this is **critical** for designing the button node's program! (BTW, there is LED button node starter code in the GitHub repo for you. Study the .h file and take advantage of the defined values and variables. **I expect you to use them in your code!**)

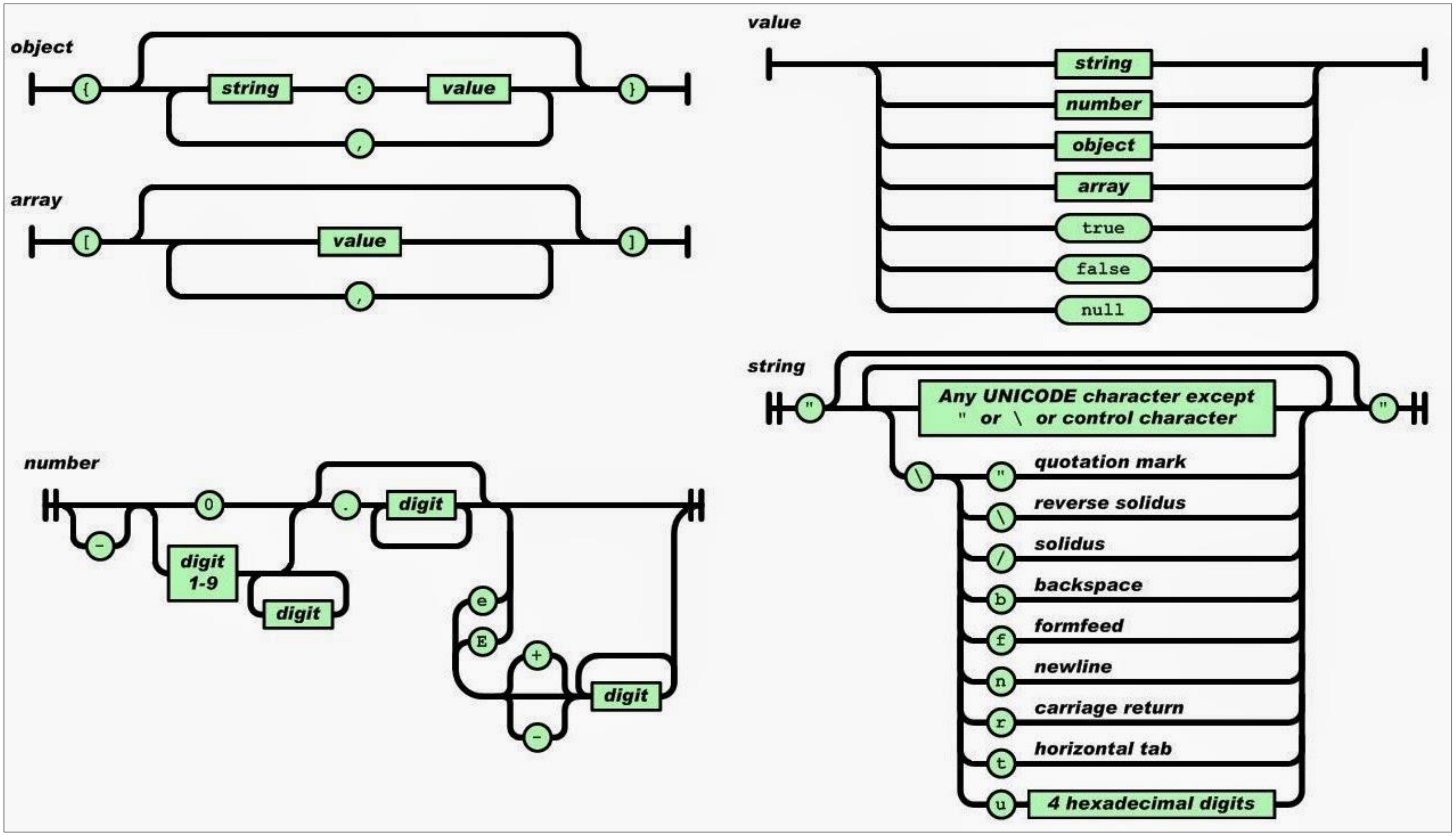
Hint: When debugging, make MQTT-Spy look like a button node by sending `ledCommand` messages and receiving `ledStatus` messages, and like an LED node by receiving `ledCommand` messages and sending `ledStatus` messages.

## Lab Report

This is a **one-week** lab. Be prepared to show your demo in lab next week. Final code must be pushed and your filled out prelab sheet need to be turned in by the deadline in canvas. **No formal lab report** is required.

# Appendix A—JSON Syntax Diagram

This diagram shows how to construct syntactically legal JSON objects:


object
{
string
:
value
,
}
array
[
value
,
]
number
-
0
digit 1-9
digit
.
digit
e
E
+
-
digit
value
string
number
object
array
true
false
null
string
"
Any UNICODE character except " or \ or control character
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits
"