Proposal On how to Handle Device Creation

TODO: expand on the following

1. Device Creator
   1. For every device we support, there should be a JSON file that acts as a template for creating a device
      1. This json file has variables that can be configured and device info (no actual value for the variables. Again, it acts as a template)
      2. DeviceId – we think it is difficult to support something like this because two of the same device types (think 2 r420s) can be in a network and it difficult to tell them apart. Instead, we think that the better solution would be to enforce a user-defined (or some default ) “nick-name” for the device so it is more easily identified
      3. How does a device interact with a network -> networkManager -> dataCollectionManager

**Introduction**

This document will detail device support in Chariot. By the nature of such systems, initial support for all devices is not possible. Thus, the aim for Chariot is to allow for developers to seamlessly integrate their devices with the system. This can be done by producing JSON template files that detail a device’s information. That is, Chariot shall allow for any device to be supported provided that a JSON file is provided to the system. The following sections will detail how to add a JSON file, how the file is used, and the reasoning behind using this method.

**Reasoning Behind Device Design**

Chariot will initially support the Impinj xArray and Imping Speedway R420. We shall support these devices, because their immediate use is beneficial to stakeholders. With only these two devices initially supported, it makes Chariot a limited system. However, the designers of the system have made efforts to design a developer-friendly approach to adding a device via a two-step system. Before listing the details of how a device is added, please refer to the Design Document to understand the architecture of Chariot. Specifically, refer to section DE14 and CD6 to understand the programming design of the Device module. As you will notice, the factory pattern is used in device creation. This is done to abstract away from device instances and allow flexibility for the system. Furthermore, notice that the all instances of devices must conform to the deviceAdapter interface. With this design, the system is flexible in device creation, but also rigid enough to ensure that devices are created under a contained design. Another key component to allow flexibility is the use of JSON files to describe a device object. These JSON files are vital for Chariot to create instances of IoT devices. Each device will have a JSON file that acts as a template. This template will be parsed by Chariot and construct an object described by the JSON file. As an example of JSON files, here is a link used that describes the Impinj xArray: <https://github.com/chariot-dev/chariot/blob/dataCollectionFramework/core/device/driver/supportedDevices.json>

**Device Support**

The first step in adding a device that Chariot supports is by producing a JSON file for that device that acts as a template. In other words, the JSON file will have a description, image link, and settings. The settings are to act as a guide for what can be altered by Chariot on a device. In other words, these settings allow Chariot to check against mismatched variable types. The reasoning behind using a JSON file to define the concept of a device is to allow flexibility for support of multiple devices.

**To Eni, this would be a good section to go into detail about the JSON files. I would touch upon the expected format.**

Once the JSON file has been defined, the template is available for Chariot to parse. The next step is to implement a class that follows the abstract class of DeviceAdapter.py. This only requires three methods to implement: connect, disconnect, and captureData. We’ve limited the it to these three methods