FIRST FRC IOT Guide

Charger Robotics - Digital Solutions

Public

Version: 1.0
Date: 3/27/2020

Table of Contents

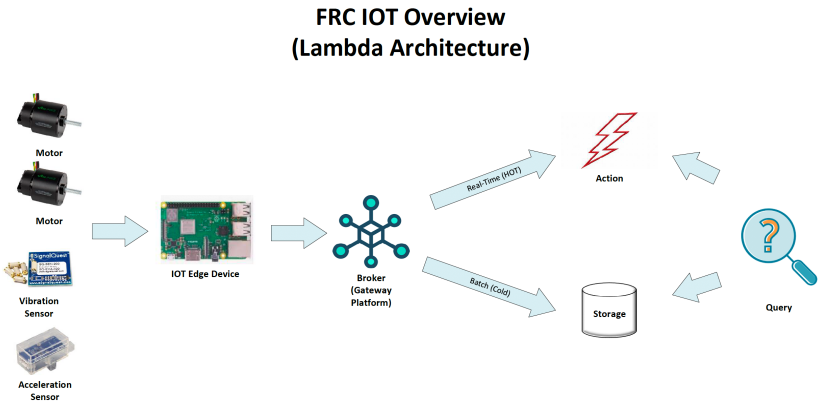
Acknowledgements 3
1. *Architecture Overview* 3
2. *Bill of Materials* 5
3. *Implementation Guide* 6
3.1. *Raspberry PI – CANBUS Edge Device* 6
3.2. *CANBUS Sampler Code* 7
3.3. *Broker API* 16
3.4. *Gateway Dashboard* 17
3.5. *Thingworx Platform Configuration* 18
3.6. *Using the Solution* 21
4. *Summary* 22
5. *FRC IOT Roadmap* 23
6. *Appendix A – Misellaneous Items* 23

Acknowledgements

Charger Robotics would like to recognize, and thank all the organizations that came together to make this initial FRC IOT release possible. To Rockwell Automation for the engineering and architecture support they provided. PTC for for their technical support, and providing the team with a Thingworx license. Also, CTR Electronics for providing a coding solution for the CANBUS Edge Device sampling. FIRST, without them none of this would be possible. To all the student participant of FIRST FRC. They are the future of STEM.

1. Architecture Overview

This FRC IOT implementation enables a speed, and a batch channel for processing sensor data from the robot. The speed channel enables stream analytics and provides real-time feedback to the FRC robot. The batch channel persists the data for further data analysis.



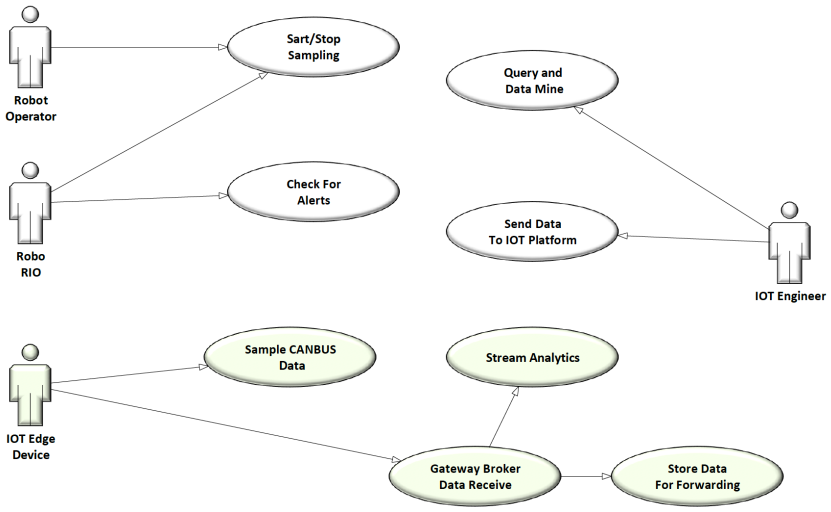
Constraints

The implementation described in this document has two constraints. Only CTR Electronics motor controllers leveraging the Phoenix SDK are supported by this solution. Also, PTC Thingworx currently does not have an educational license for an on-prem (FRC Driver's Console) deployment. This implementation leverages a PaaS cloud offering by PTC. This limits the real-time capabilities of the Thingworx platform. The solution design will change in future when Thingworx educational licenses become available for on-prem.

The robot network needs be configured to use static IP addresses. See the Appendix at the end of this paper for more information.

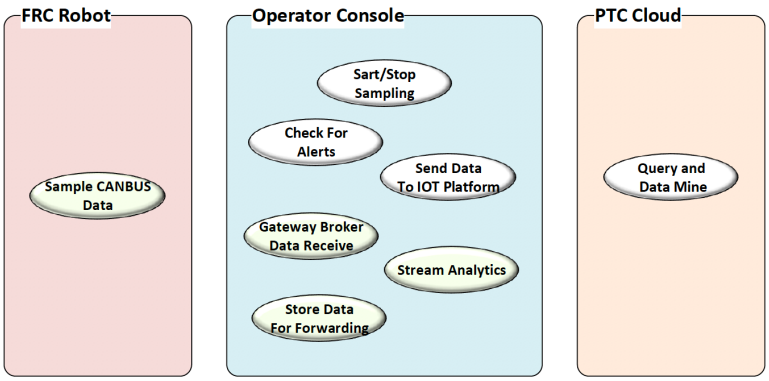
Use Cases

The following use-cases are involved in the solution.



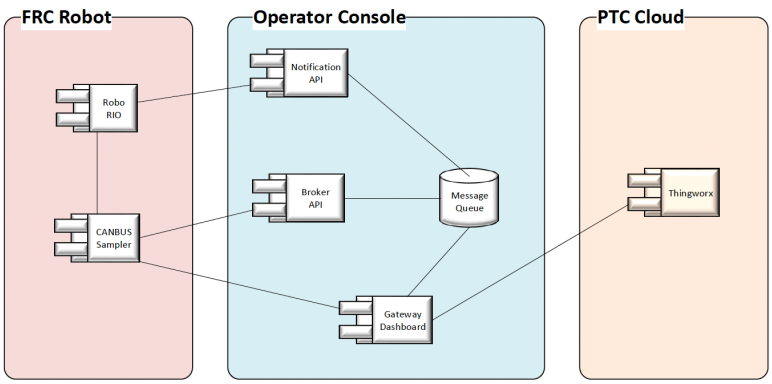
Use Cases Mapping

The following diagram describes where in the eco-system the use cases are implemented.



Components and Deployment

The following diagram describes where in the components are deployed.

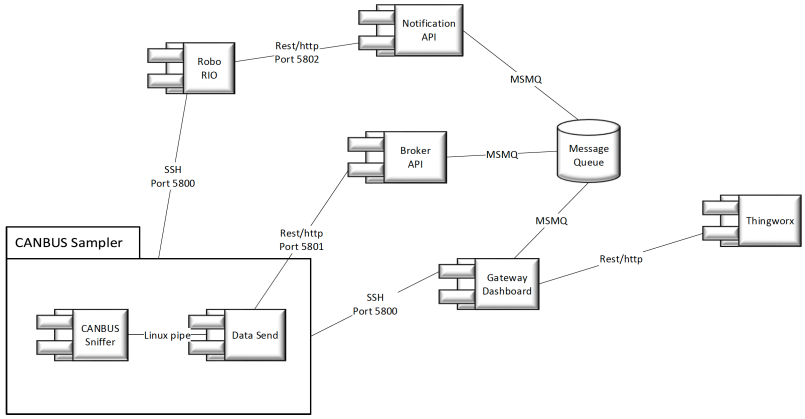


Component to Technology Mapping

Component	Technology
CANBUS Sniffer	C++
Data Sender	Java
CANBUS Sampler	Linux Script
Broker API	ASP.net
Gateway Dashboard	C# Forms
Notification API	ASP.net
Datastore	MSMQ

Protocol Mapping

This diagram documents the protocols and ports used for communication between components.



2. Bill of Materials

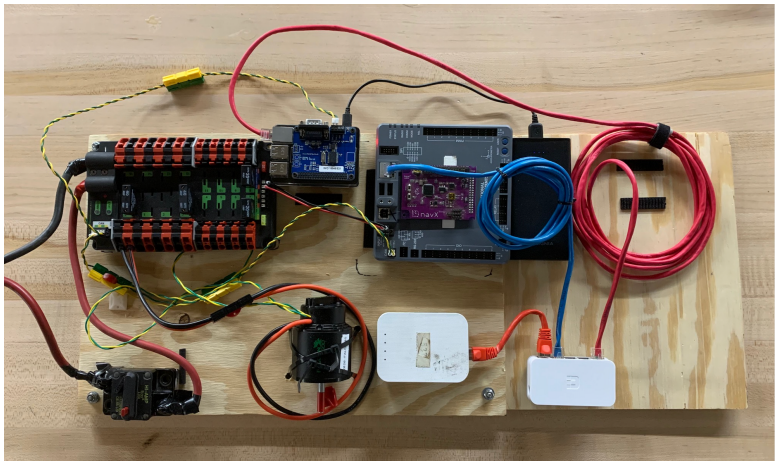
The following materials are required to build an FRC IOT solution.

- Raspberry Pi 3 B+ System With Single CAN Bus Interface. https://copperhilltech.com/raspberry-pi-3-b-system-with-single-can-bus-interface/
- PTC Thingworx Educational License:
- Network Hub/Switch.
- Talon SRX Motor Controller (any Phoenix code based controller)
- RoboRIO
- CAN Bus Controller
- FRC Radio

- Windows 10 PC as Driver's Console
- USB Lithium Battery

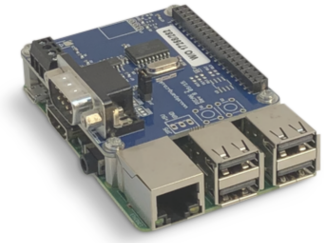
3. Implementation Guide

This section provides instructions for implementing an IOT solution for a FIRST FRC Robot. The implementation pulls from a few external sources. The instructions will reference those sources where applicable. Below is a picture of the testboard configuration.



3.1. Raspberry PI – CANBUS Edge Device

A CANBUS Edge Device is required for the sampling of motor controller data. The PI will have it's own power source supplied by a USB lithium battery (check FRC rules for allowed specs). While the sampling could be done on the RoboRIO, it is preferred to move the IOT processing to a separate device. This will prevent the sampling code from interfering with robot operations. The Edge Device is a Raspberry Pi with a Control Area Network (CAN) Interface. The Raspberry PI with single can-bus interface from Copperhill Technologies is recommended: https://copperhilltech.com/raspberry-pi-3-b-system-with-single-can-bus-interface/



Follow the initial configuration instructions provided on the Copperhill website. https://copperhilltech.com/pican2-controller-area-network-can-interface-for-raspberry-pi/ For this implementation you will want to use the screw terminal to connect to the CANBUS network. The CANBUS interface on the PI terminates the CANBUS network, so the Pi will need to be at the end

of the network and the PDP on the robot will need to be configured not to terminate. The BAUD rate needs to be set to 1000000 to communicate with the Phoenix SDK. You will also want to configure the PI to have a static IP of 10.xx.xx.222.

3.2. CANBUS Sampler Code

The code to sample the motor controller data from the CANBUS will run on the Raspberry PI configured in secton 3.1. The Sample is made up of 3 components; a C++ code, Java code, and Linux Script. The C++ code samples the motor control data that is then piped to a Jave program that sends the data to the Operator Console. The long term goal is to combine these into one C++ or Java program (see FRC IOT Roadmap section).

C++ Code

The Phoenix C++ libraries are required for communication with the motor controllers. CTR Electronics provides an example C++ source code for working with the Phoenix based controllers. The repository for the project is located here: https://github.com/CrossTheRoadElec/Phoenix-Linux-SocketCAN-Example The code in the example.cpp file needs to be replaced with the eample.cpp code located here: https://github.com/beckologist/FRCIOT. This code creates a Talon object, smaples the motor controller data, and writes the data to standard out in a JSON format. The output message is prefeced with a command code of “payload::”. This will be used by the Java program to identify sampling messages from other messages sent to standard out. You can test the C++ program standalone to validate that it is connecting to the motor controller and sampling the data. Note that the progrs up to 8 ommand line arguments that specify which motors are to be sampled.

```
#define Phoenix_No_WPI // remove WPI dependencies
  #include "ctre/Phoenix.h"
  #include "ctre/phoenix/platform/Platform.h"
  #include "ctre/phoenix/unmanaged/Unmanaged.h"
  #include <string>
  #include <iostream>
  #include <fstream>
  #include <chrono>
  #include <thread>
  /*#include <SDL2/SDL.h>*/
  #include <unistd.h>
  #include <chrono>
  #include <sys/socket.h>
  #include <netinet/in.h>
  #include <netdb.h>
  #include <stdlib.h>
  #include <stdio.h>
  #include <string.h>
  #include <time.h>
  #include <ctime>
  #include <iomanip>
  #include <sstream>


  using namespace ctre::phoenix;
  using namespace ctre::phoenix::platform;
  using namespace ctre::phoenix::motorcontrol;
  using namespace ctre::phoenix::motorcontrol::can;
  using namespace std;

  /* make some talons for drive train */
  //TalonSRX theTalon1(1);
  //TalonSRX theTalon2(2);
  //TalonSRX theTalon3(3);
  //TalonSRX theTalon4(4);
  //TalonSRX theTalon5(5);
  //TalonSRX theTalon6(6);
  //TalonSRX theTalon7(7);
  //TalonSRX theTalon8(8);
```

```
void thread1()
{
	TalonSRX theTalon1(1);

	std::ostringstream oss;

	string myPayload = "";

	while (true) {

		//Talon 1
		myPayload = "";

		oss.str("");
		auto now = std::chrono::system_clock::now();
		auto time = std::chrono::system_clock::to_time_t(now);
		auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
			std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

		oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
		oss << ms.count();
		auto str = oss.str();
		string myTimeStamp = str;

		myPayload = myPayload + "payload::";
		myPayload = myPayload + "537_Motor1*";
		myPayload = myPayload + "{";
		myPayload = myPayload + "\"TimeStamp\": ";
		myPayload = myPayload + "\"" + myTimeStamp + "\"";
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"Temperature\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetTemperature());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"OutputCurrent\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetOutputCurrent());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"MotorOutputPercent\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetMotorOutputPercent());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"MotorOutputVoltage\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetMotorOutputVoltage());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"SensorPosition\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetSelectedSensorPosition());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"SensorVelocity\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetSelectedSensorVelocity());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetActiveTrajectoryArbFeedFwd());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetActiveTrajectoryPosition());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetActiveTrajectoryVelocity());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"BusVoltage\": ";
		myPayload = myPayload + std::to_string(theTalon1.GetBusVoltage());
		myPayload = myPayload + "}";

		myPayload = myPayload + "payload End::";

		std::cout << myPayload;
	}//while end
}//thread1


void thread2()
{

	TalonSRX theTalon2(2);

	std::ostringstream oss;

	string myPayload = "";

	while (true) {
		//Talon 2
		myPayload = "";

		oss.str("");
		auto now = std::chrono::system_clock::now();
		auto time = std::chrono::system_clock::to_time_t(now);
		auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
			std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());
```

```
			oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
			oss << ms.count();
			auto str = oss.str();
			string myTimeStamp = str;

			myPayload = myPayload + "payload::";
			myPayload = myPayload + "537_Motor2*";
			myPayload = myPayload + "{";
			myPayload = myPayload + "\"TimeStamp\": ";
			myPayload = myPayload + "\"" + myTimeStamp + "\"";
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"Temperature\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetTemperature());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"OutputCurrent\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetOutputCurrent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputPercent\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetMotorOutputPercent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetMotorOutputVoltage());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorPosition\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetSelectedSensorPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetSelectedSensorVelocity());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetActiveTrajectoryArbFeedFwd());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetActiveTrajectoryPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetActiveTrajectoryVelocity());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"BusVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon2.GetBusVoltage());
			myPayload = myPayload + "}";

			myPayload = myPayload + "payload End::";

			std::cout << myPayload;
		}//while end
}//thread2

void thread3()
{
		TalonSRX theTalon3(3);

		std::ostringstream oss;

		string myPayload = "";

		while (true) {
			//Talon 3
			myPayload = "";

			oss.str("");
			auto now = std::chrono::system_clock::now();
			auto time = std::chrono::system_clock::to_time_t(now);
			auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
				std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

			oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
			oss << ms.count();
			auto str = oss.str();
			string myTimeStamp = str;

			myPayload = myPayload + "payload::";
			myPayload = myPayload + "537_Motor3*";
			myPayload = myPayload + "{";
			myPayload = myPayload + "\"TimeStamp\": ";
			myPayload = myPayload + "\"" + myTimeStamp + "\"";
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"Temperature\": ";
			myPayload = myPayload + std::to_string(theTalon3.GetTemperature());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"OutputCurrent\": ";
			myPayload = myPayload + std::to_string(theTalon3.GetOutputCurrent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputPercent\": ";
			myPayload = myPayload + std::to_string(theTalon3.GetMotorOutputPercent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon3.GetMotorOutputVoltage());
```

```
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorPosition\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetSelectedSensorPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetSelectedSensorVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetActiveTrajectoryArbFeedFwd());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetActiveTrajectoryPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetActiveTrajectoryVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"BusVoltage\": ";
                myPayload = myPayload + std::to_string(theTalon3.GetBusVoltage());
                myPayload = myPayload + "}";

                myPayload = myPayload + "payload End::";

                std::cout << myPayload;
        }//while end
}//thread3


void thread4()
{

        TalonSRX theTalon4(4);

        std::ostringstream oss;

        string myPayload = "";

        while (true) {
                //Talon 4
                myPayload = "";

                oss.str("");
                auto now = std::chrono::system_clock::now();
                auto time = std::chrono::system_clock::to_time_t(now);
                auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
                        std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

                oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
                oss << ms.count();
                auto str = oss.str();
                string myTimeStamp = str;

                myPayload = myPayload + "payload::";
                myPayload = myPayload + "537_Motor4*";
                myPayload = myPayload + "{";
                myPayload = myPayload + "\"TimeStamp\": ";
                myPayload = myPayload + "\"" + myTimeStamp + "\"";
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"Temperature\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetTemperature());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"OutputCurrent\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetOutputCurrent());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"MotorOutputPercent\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetMotorOutputPercent());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"MotorOutputVoltage\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetMotorOutputVoltage());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorPosition\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetSelectedSensorPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetSelectedSensorVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetActiveTrajectoryArbFeedFwd());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetActiveTrajectoryPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetActiveTrajectoryVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"BusVoltage\": ";
                myPayload = myPayload + std::to_string(theTalon4.GetBusVoltage());
                myPayload = myPayload + "}";

                myPayload = myPayload + "payload End::";
```

```
		std::cout << myPayload;
	}//while end
}//thread4


void thread5()
{
	TalonSRX theTalon5(5);

	std::ostringstream oss;

	string myPayload = "";

	while (true) {
		//Talon 5
		myPayload = "";

		oss.str("");
		auto now = std::chrono::system_clock::now();
		auto time = std::chrono::system_clock::to_time_t(now);
		auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
			std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

		oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
		oss << ms.count();
		auto str = oss.str();
		string myTimeStamp = str;

		myPayload = myPayload + "payload::";
		myPayload = myPayload + "537_Motor5*";
		myPayload = myPayload + "{";
		myPayload = myPayload + "\"TimeStamp\": ";
		myPayload = myPayload + "\"" + myTimeStamp + "\"";
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"Temperature\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetTemperature());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"OutputCurrent\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetOutputCurrent());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"MotorOutputPercent\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetMotorOutputPercent());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"MotorOutputVoltage\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetMotorOutputVoltage());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"SensorPosition\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetSelectedSensorPosition());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"SensorVelocity\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetSelectedSensorVelocity());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetActiveTrajectoryArbFeedFwd());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetActiveTrajectoryPosition());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetActiveTrajectoryVelocity());
		myPayload = myPayload + ",";
		myPayload = myPayload + "\"BusVoltage\": ";
		myPayload = myPayload + std::to_string(theTalon5.GetBusVoltage());
		myPayload = myPayload + "}";

		myPayload = myPayload + "payload End::";

		std::cout << myPayload;
	}//while end
}//thread5


void thread6()
{
	TalonSRX theTalon6(6);

	std::ostringstream oss;

	string myPayload = "";

	while (true) {
		//Talon 6
		myPayload = "";
		oss.str("");
		auto now = std::chrono::system_clock::now();
```

```
                auto time = std::chrono::system_clock::to_time_t(now);
                auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
                        std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

                oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
                oss << ms.count();
                auto str = oss.str();
                string myTimeStamp = str;

                myPayload = myPayload + "payload::";
                myPayload = myPayload + "537_Motor6*";
                myPayload = myPayload + "{";
                myPayload = myPayload + "\"TimeStamp\": ";
                myPayload = myPayload + "\"" + myTimeStamp + "\"";
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"Temperature\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetTemperature());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"OutputCurrent\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetOutputCurrent());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"MotorOutputPercent\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetMotorOutputPercent());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"MotorOutputVoltage\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetMotorOutputVoltage());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorPosition\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetSelectedSensorPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"SensorVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetSelectedSensorVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetActiveTrajectoryArbFeedFwd());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetActiveTrajectoryPosition());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetActiveTrajectoryVelocity());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"BusVoltage\": ";
                myPayload = myPayload + std::to_string(theTalon6.GetBusVoltage());
                myPayload = myPayload + "}";

                myPayload = myPayload + "payload End::";

                std::cout << myPayload;
        }//while end
}//thread6


void thread7()
{

        TalonSRX theTalon7(7);

        std::ostringstream oss;

        string myPayload = "";

        while (true) {
                //Talon 7
                myPayload = "";
                oss.str("");
                auto now = std::chrono::system_clock::now();
                auto time = std::chrono::system_clock::to_time_t(now);
                auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
                        std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

                oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
                oss << ms.count();
                auto str = oss.str();
                string myTimeStamp = str;

                myPayload = myPayload + "payload::";
                myPayload = myPayload + "537_Motor7*";
                myPayload = myPayload + "{";
                myPayload = myPayload + "\"TimeStamp\": ";
                myPayload = myPayload + "\"" + myTimeStamp + "\"";
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"Temperature\": ";
                myPayload = myPayload + std::to_string(theTalon7.GetTemperature());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"OutputCurrent\": ";
                myPayload = myPayload + std::to_string(theTalon7.GetOutputCurrent());
                myPayload = myPayload + ",";
                myPayload = myPayload + "\"MotorOutputPercent\": ";
```

```
			myPayload = myPayload + std::to_string(theTalon7.GetMotorOutputPercent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetMotorOutputVoltage());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorPosition\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetSelectedSensorPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetSelectedSensorVelocity());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetActiveTrajectoryArbFeedFwd());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetActiveTrajectoryPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetActiveTrajectoryVelocity());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"BusVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon7.GetBusVoltage());
			myPayload = myPayload + "}";

			myPayload = myPayload + "payload End::";

			std::cout << myPayload;
		}//while end
}//thread7



void thread8()
{
		TalonSRX theTalon8(8);

		std::ostringstream oss;

		string myPayload = "";

		while (true) {
			//Talon 8
			myPayload = "";

			oss.str("");
			auto now = std::chrono::system_clock::now();
			auto time = std::chrono::system_clock::to_time_t(now);
			auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(now.time_since_epoch()) -
				std::chrono::duration_cast<std::chrono::seconds>(now.time_since_epoch());

			oss << std::put_time(std::localtime(&time), "%Y-%m-%d") << "T" << std::put_time(std::localtime(&time), "%H:%M:%S.");
			oss << ms.count();
			auto str = oss.str();
			string myTimeStamp = str;

			myPayload = myPayload + "payload::";
			myPayload = myPayload + "537_Motor8*";
			myPayload = myPayload + "{";
			myPayload = myPayload + "\"TimeStamp\": ";
			myPayload = myPayload + "\"" + myTimeStamp + "\"";
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"Temperature\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetTemperature());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"OutputCurrent\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetOutputCurrent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputPercent\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetMotorOutputPercent());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"MotorOutputVoltage\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetMotorOutputVoltage());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorPosition\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetSelectedSensorPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"SensorVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetSelectedSensorVelocity());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryArbFeedFwd\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetActiveTrajectoryArbFeedFwd());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"GetActiveTrajectoryPosition\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetActiveTrajectoryPosition());
			myPayload = myPayload + ",";
			myPayload = myPayload + "\"ActiveTrajectoryVelocity\": ";
			myPayload = myPayload + std::to_string(theTalon8.GetActiveTrajectoryVelocity());
			myPayload = myPayload + ",";
```

```
                        myPayload = myPayload + "\"BusVoltage\": ";
                        myPayload = myPayload + std::to_string(theTalon8.GetBusVoltage());
                        myPayload = myPayload + "}";

                        myPayload = myPayload + "payload End::";

                        std::cout << myPayload;
                }//while end
        }//thread8


        int main(int argc, char** argv) {

                for (int i = 0; i < argc; i++)
                {

                        string myArg = argv[i];

                        if (myArg == "1"){
                                thread th1(thread1);
                                th1.detach();
                        }

                        if (myArg == "2"){
                                thread th2(thread2);
                                th2.detach();
                        }

                        if (myArg == "3"){
                                thread th3(thread3);
                                th3.detach();
                        }

                        if (myArg == "4"){
                                thread th4(thread4);
                                th4.detach();
                        }

                        if (myArg == "5"){
                                thread th5(thread5);
                                th5.detach();
                        }

                        if (myArg == "6"){
                                thread th6(thread6);
                                th6.detach();
                        }

                        if (myArg == "7"){
                                thread th7(thread7);
                                th7.detach();
                        }

                        if (myArg == "8"){
                                thread th8(thread8);
                                th8.detach();
                        }

                }//for end


                while (true)
                {
                }

}
```

Java Code

The java code reads data from std in, cecks that it is a sampling message, then send the data to the Operator Console. Create a Java program named “ThingworxSend” with the code shown below. A copy of the code is located here: https://github.com/beckologist/FRCIOT. Note that the highlighted code will need to change to match your FRC Team number. This code takes two commandline arguments for the IP address, and Port Number, of the Broker API

```
/**
 * Write a description of class ThingworxSend here.
 *
 * @author (David Kaiser)
```

```
 * @version (V1.0.0)
 */

import java.io.*;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;


public class ThingworxSend
{

   public static void main(String[] args)
   {

      InputStreamReader isReader = new InputStreamReader(System.in);
      BufferedReader bufReader = new BufferedReader(isReader);

      String inputStr = "";

         try{
            while(inputStr != null){

               String output = "xxx";
               inputStr=bufReader.readLine();

               if(inputStr != null && (inputStr.length() > 9)){

                  String myMessageType = inputStr.substring(0,9);

                  if(myMessageType.equals("payload::")){

                     int myEndParseIndex = inputStr.indexOf("payload End::");
                     output = (inputStr.substring(9,myEndParseIndex));

                     //Http Send
                     String myURLString = "http://" + args[0] + ":" + args[1] + "/api/537/Things/Data";
                     //URL url = new URL("http://10.5.37.201:5801/api/537/Things/Data");
                     URL url = new URL(myURLString);

                     HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                     conn.setDoOutput(true);
                     conn.setRequestMethod("PUT");
                     conn.setRequestProperty("Content-Type", "application/json");
                     //conn.setRequestProperty("appKey", "b25c870b-1d4e-4f19-978e-2521bdb1929f");

                     OutputStreamWriter os = new OutputStreamWriter(conn.getOutputStream());
                     os.write(output);
                     os.close();
                     conn.getInputStream();
                     System.out.println(conn.getResponseCode());

                  }
                  else{
                     System.out.println(inputStr);
                  }

               } else{
                   System.out.println("input string is null");
               }
          }
         }
         catch(Exception e){
             System.out.println(e);
            e.printStackTrace();
         }

      }

   }
```

Script

Create a script named “sample.sh with the following code. A copy of the code is located here: https://github.com/beckologist/FRCIOT Make this script executable using the chmod -x command. This will be the script called from the Gateway Dashboard to start and stop the sampling. This script takes several command line arguments. The first 3 are required args. They are: ***Broker API IP Address, Broker API Port, Broker API Key***. After that there 8 optional args. These optional args are the ID numbers of the motoers to be sampled.

```
# Save args to variables
myIP=$1
```

```
shift
myPort=$1
shift
myKey=$1
shift
m1=$1
shift
m2=$1
shift
m3=$1
shift
m4=$1
shift
m5=$1
shift
m6=$1
shift
m7=$1
shift
m8=$1

#Call the sampling code and pipe to the sending code
./example $m1 $m2 $m3 $m4 $m5 $m6 $m7 $m8 | java -cp ./  ThingworxSend $myIP $myPort $myKey
```

Deployment

Deploy the C++ executable (example), Java executable (ThingworxSend.class), and the sample.sh script into the root directory of the Raspberry PI. The Gateway Dashboard will be looking to this directory for these files.

3.3. Broker API

The broker API is the listener on the Operators Station (Windows 10 PC) that will receive data from the Java program of the CANBUS Sampler. This is a self-hosted ASP.net Restful API. When the Broker API receives data, it writes that data to an inbound queue for processing by the Gateway Dashboard. When the Broker API is started, it will check if the appropriate queues exist. If they do not exist, the Brocker API will create them.

The source code for the Broker API can be found here: https://github.com/beckologist/GatewayAPI

Compile the code and run the program (may need to run as “Administrator”). You should see the following results:

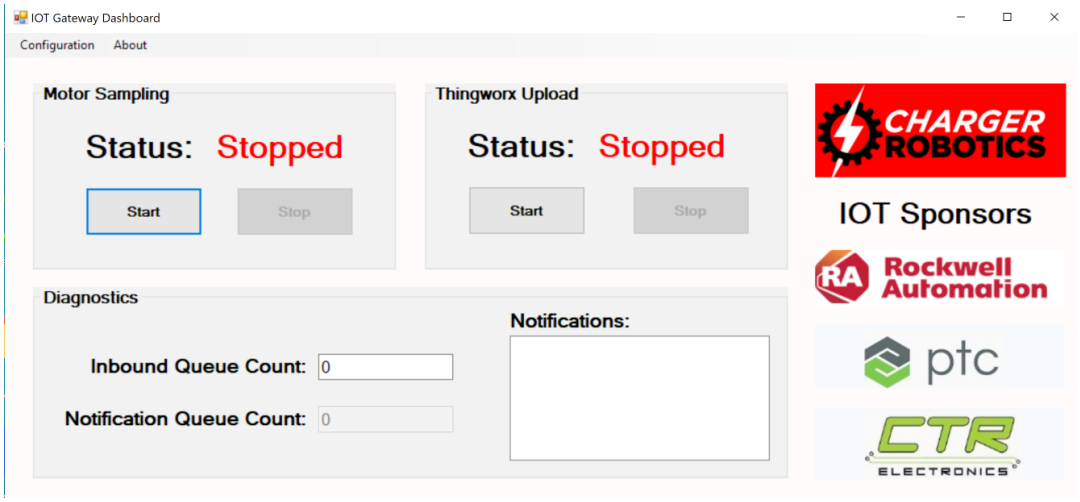
Note: The Windows 10 firewall rules will need to be configured to allow external HTTP calls to port 5801. Depending on your teams security posture, you may want to only allow the IP address of the CANBUS Sample to access this Port.

3.4. Gateway Dashboard

The Gateway Dashboard is the UI for controlling the FRC IOT eco-system. It provides the following capabilities:

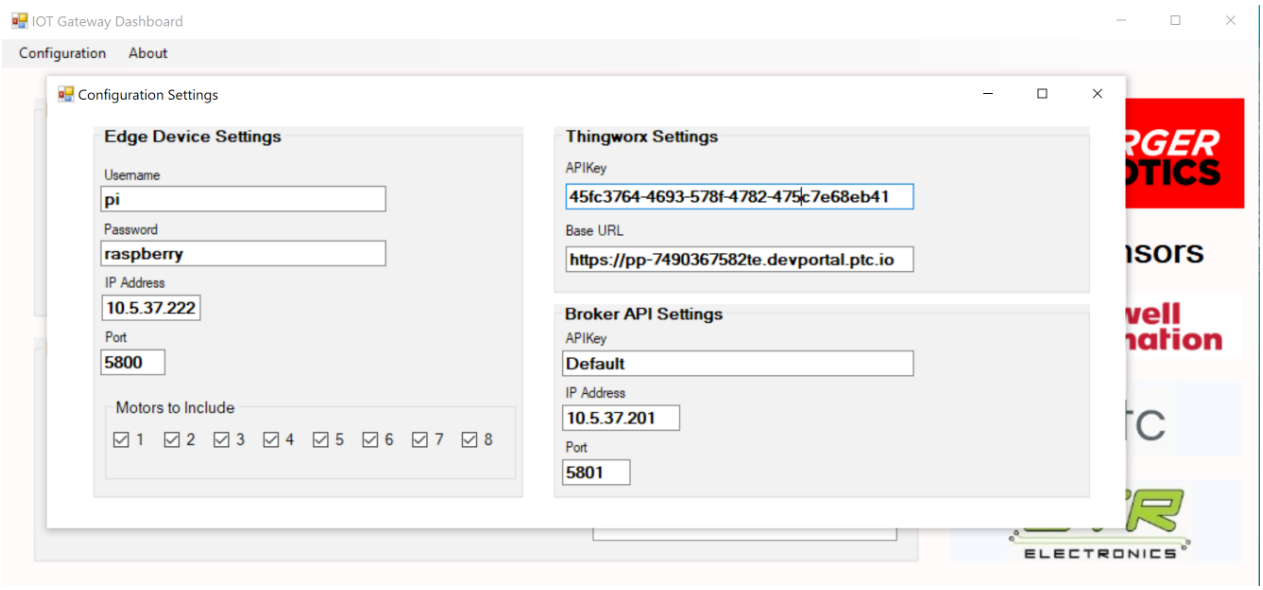
- Start/Stop sampling and sending of the CANBUS motor controller data
- Visability into the queueing activity
- Start/Stop of data forwarding to the Thingworx Cloud Platform
- Visability of Notifiation messages for the RoboRIO
- Configuration of system parameters

The source code for the Gateway Dashboard can be found here: https://github.com/beckologist/GatewayDashboard. Compile the code and run the program (may need to run as “Administrator”). You should see the following results:



3.4.1. System Configuration Parameters

The parameters for configuring the IOT system can be foud under the Configuration -> Setting menu.



3.5. Thingworx Platform Configuration

The final component to configure is the Thingworx platform. There are several options available for Thingworx deployment. At the time of this writing, an educational license did not exist for anything but a Cloud deployment.

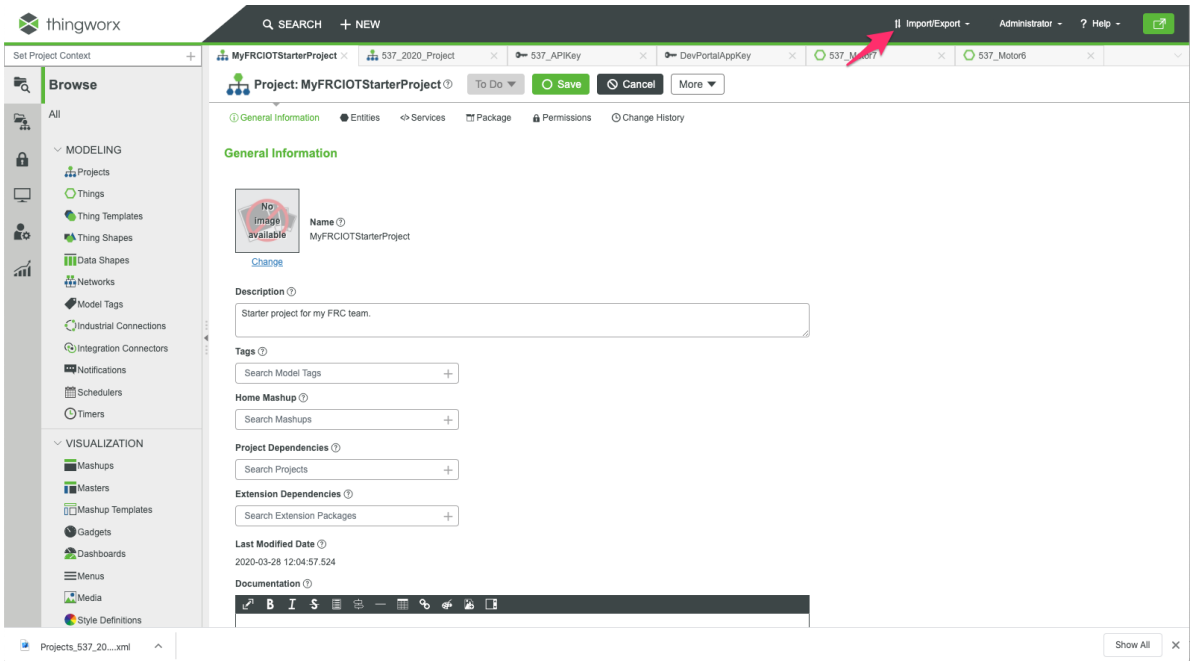
Platform Installation (Windows)
https://developer.thingworx.com/en/resources/guides/foundation-windows-install-guide

Platform Configuration
https://developer.thingworx.com/resources/guides/thingworx-foundation-quickstart

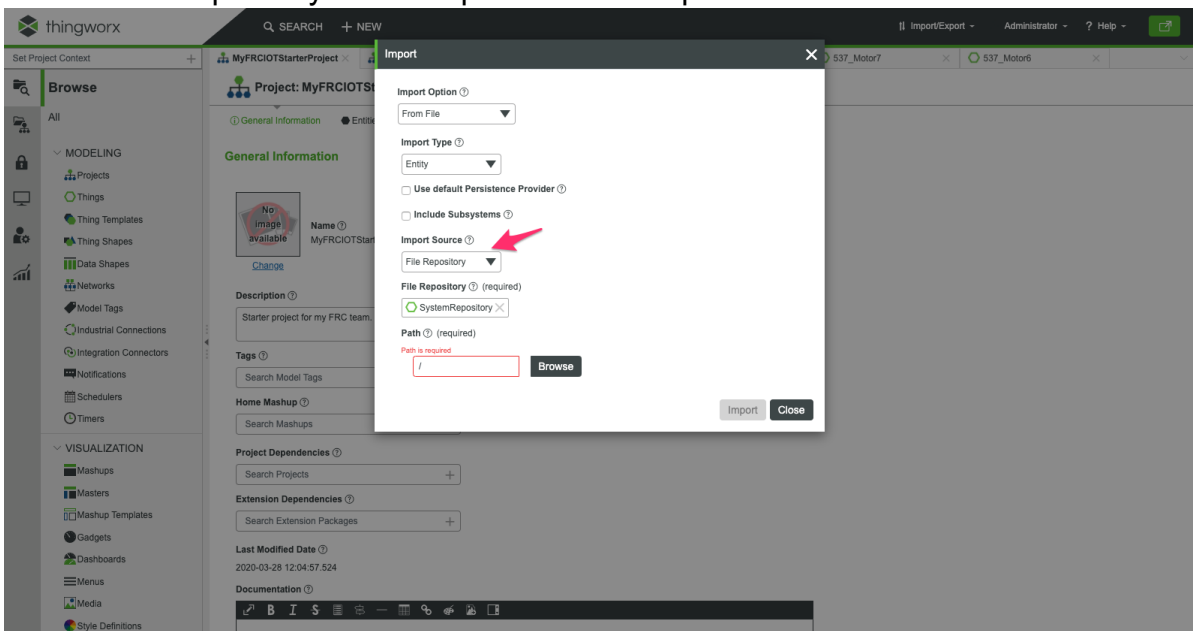
Once the Thingworx instance is up and running, a “Starter” project configuration file can be imported into Thingworx. This is an XML file that contains a prebuilt project for working with up to 8 motors. Ii provides all the required Thing Shapes, Thing Templates, and Things for up to 8 motors. A set of basic Mashups are also provided as a starting point.

The name of the file is FRC_IOT_Starter_Project.xml. The file is located here: https://github.com/beckologist/FRCIOT. Once you are logged into Thingworx:

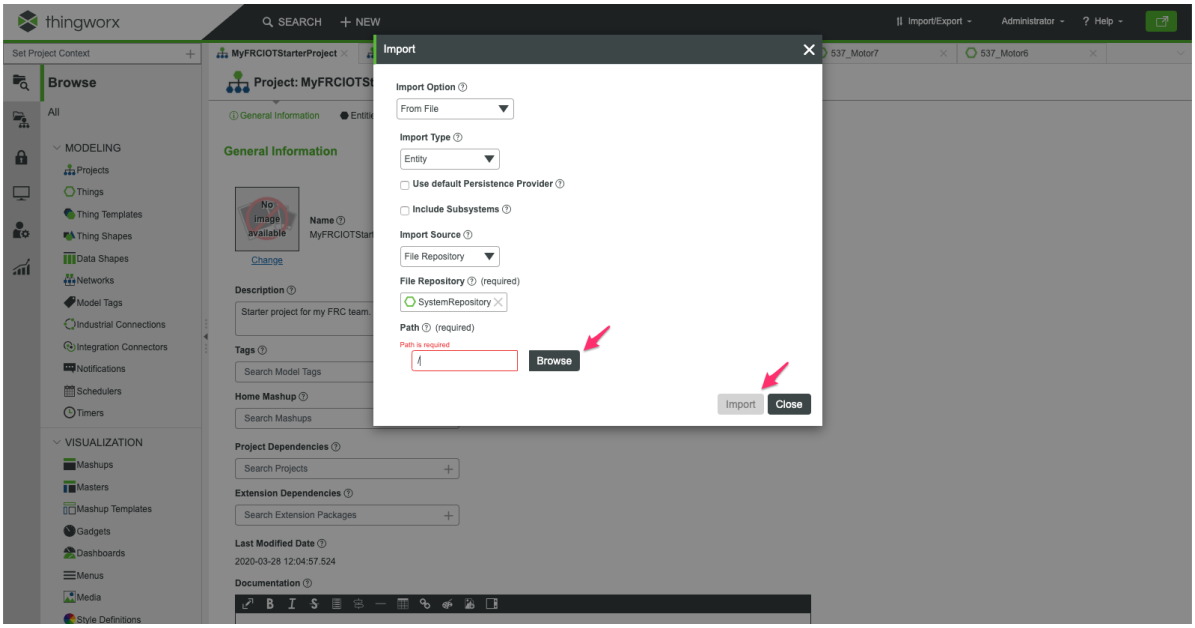
1. Select Import/Export at the top of the screen.



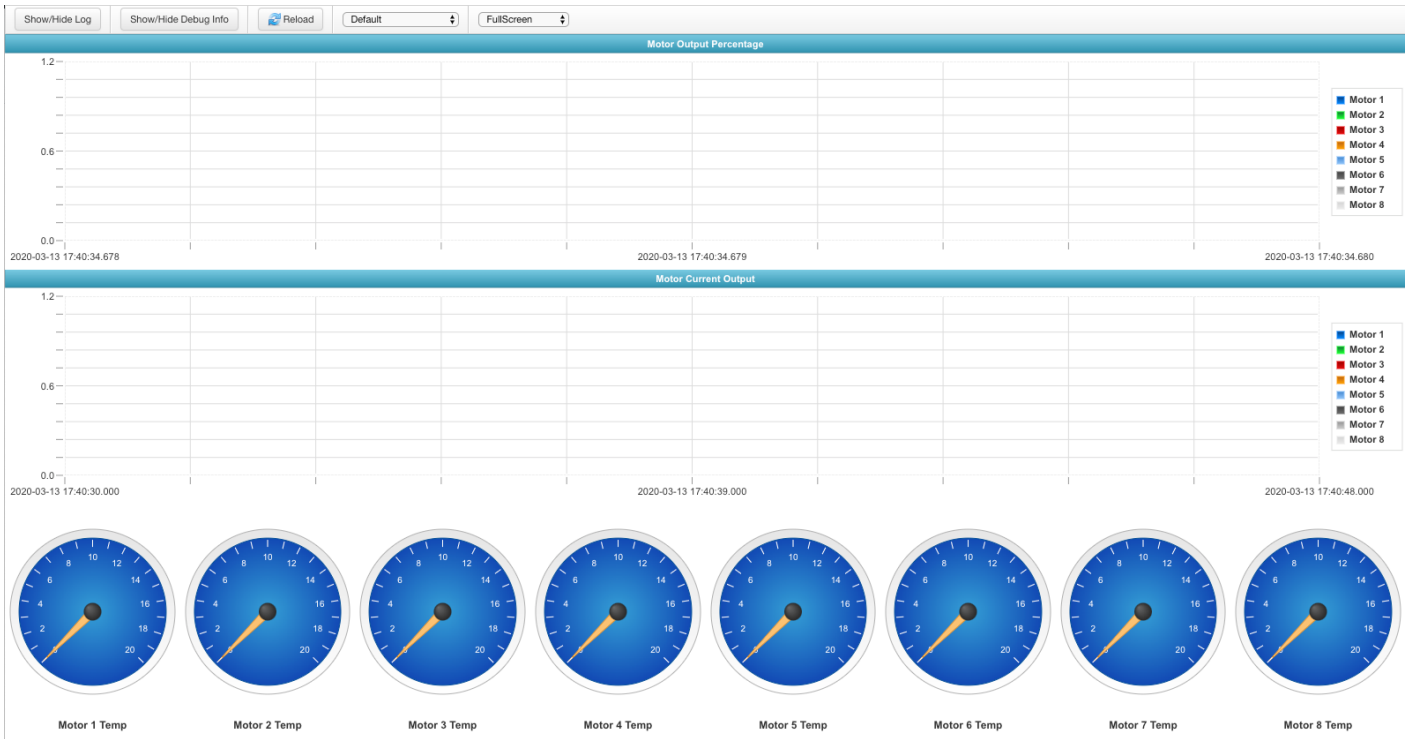
2. Select “File Repository” from Import Source drop down.



3. Browse to the FRC_IOT_Starter_Project.xml file that was downloaded from the GITHUB repository(https://github.com/beckologist/FRCIOT). Then click on the Import button.



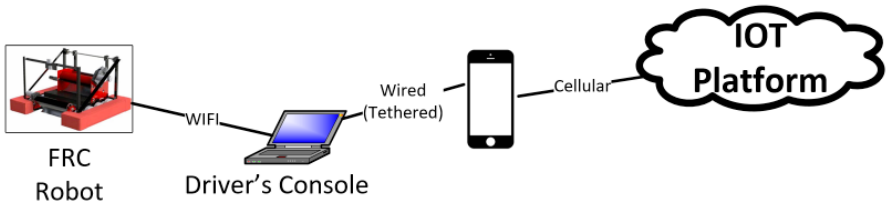
Standard Mashups are provided in the FRC_IOT_Starter_Project.xml file.



3.6. Using the Solution

There are two different scenarios that an FRC IOT solution will be used; Competition, and Non-Competition. Per FRC rules state that during competion the Driver's Console can only be connected to field network. No other netrok connections are permitted. If your Thingworx instance is not running locally on the Driver's Console (i.e. a cloud-hosted instance), the IOT data must be stored locally on the Driver's Console, and forwarded to a Thingworx Cloud instance after the match has completed. Fortunately, the Gateway Dashboard supports all possible Thingworx deployment options.

The Driver's Console running the Broker API and Gateway Dashboard will need to be able to connect to the FRC robot radio, and the internet. This will require the standard FRC Driver's Console connection to the FRC Radio on the robot, and a tethered cellular device for internet access.



Checklist:

- The CANBUS Edge Device is configured a ready to start sending data (Section 3.2).
- The Broker API is deployed and running with the appropriate firewall settings (Section 3.3).
- The Gateway Dashboard is installed,configured, and ready to use (Section 3.4).
- A Thingworx instance is configured and ready to use (Section 3.5).

3.6.1. Non-Competition (Cloud-hosted Thingworx)

1. Connect the Driver's Console the the FRC Radio on the robot via WIFI.
2. Teather the Driver's Station to a cellphone or tablet with celluar data connection.
3. Click on the "Start" button in the Motor Sampling section of the Gateway Dashboard. This will ssh to the CANBUS Edge Device and start the sample.sh script. The Inbound Queue Count numbers should begin to go up as messages arrive. Messages are stored in the MSMQ queues for forwarding when requested.
4. Click on the "Start" button in the Thingworx Upload section of the Gateway Dashboard. This will start the sending of messages to the Thingworx platform.

This will result in the data being sent to Thingworx in "realtime".

3.6.2. Competition (Cloud-hosted Thingworx)

1. Connect the Driver's Console the the FRC Radio on the robot via WIFI.
2. Click on the "Start" button in the Motor Sampling section of the Gateway Dashboard. This will ssh to the CANBUS Edge Device and start the sample.sh script. The Inbound Queue Count numbers should begin to go up as messages arrive. Messages are stored in the MSMQ queues for forwarding when requested.

***** Compete in the competion round – drive it like you stole it! *****
Once your Alliance has won, and you are back in the Pit Area…

3. Teather the Driver's Station to a cellphone or tablet with celluar data connection.
4. Click on the "Start" button in the Thingworx Upload section of the Gateway Dashboard. This will start the sending of messages to the Thingworx platform.

This will result in the data being sent to Thingworx.

3.6.3. Competition (Locally-hosted Thingworx)

This is similar to Non-Competition(Cloud-hosted Thingworx), with the exception of not needing a Cellular connection.

1. Connect the Driver's Console the the FRC Radio on the robot via WIFI.
2. Click on the "Start" button in the Motor Sampling section of the Gateway Dashboard. This will ssh to the CANBUS Edge Device and start the sample.sh script. The Inbound Queue Count numbers should begin to go up as messages arrive. Messages are stored in the MSMQ queues for forwarding when requested.
3. Click on the "Start" button in the Thingworx Upload section of the Gateway Dashboard. This will start the sending of messages to the Thingworx platform.

This will result in the data being sent to Thingworx in "realtime".

4. Summary

This FRC IOT Guide provided all the information needed to get started with IOT for an FRC robot. It documents an MVP (Minimally Viable Product) implementation for FRC Team 537- Charger Robotics. Some of the implementation choices were a compromise due to licensing and/or time constraints for the first release of an MVP. Section 5 defines a roadmap for future capabilities, and refinment of existing capabilities.

5. FRC IOT Roadmap

	Capability	Dependancy	Interation 1	Interation 2	Interation 3	Interation 4
1	Feedback Notification					
2	Convert Sampler Code to all Java	Ex1				
3	Migrate Thingworx to Operator Console	Ex2				
4	Add Vuforia AR	EX3				
5	Convert Sampler Code sends to send directly to Thingworx	3				
6	Use Thingworks Feedback Notification	3				
7	Add input parm for number of motors to sample					

External Dependencies

EX1 CTRE Java SDK for Raspberry PI

EX2 PTC to offer an Educational offering to run on Windows platform

EX3 PTC to offer an ed. license for vuforia, or a university partner

6. Appendix A – Misellaneous Items

- CAN Interface Baudrate = 1 000 000
- Static IPs
- Set PI SSH port to 5800
- Broker API listens on port 5801
- Notification API listens on 5802
- PI Static IP is 10.5.3.222
- Gateway inbound queue named: IOTData
- Gateway notification queue named: IOTNotification