iRduino Application

Basic Structure of Solution

- **ArduinoInterfaces**

Classes for direct communication and handling of the Arduino. Basically, the simple sending and receiving of messages along the serial port

- **iRacingSDKWrapper**

The SDK wrapper that the application uses to communicate and get data from the iRacing API. This was not developed by me but is included in the solution because it has been modified to add new functions but mainly allow more variables to be retrieved from the API

- **iRduino**

Main application project. Set this as the Startup project.

- **iRSDKSharp**

The SDK wrapper using this API interface. Has a few small modifications. Not developed by me either.

- **ShiftData**

Contains the shift light data for each car. It allows for custom shift lights for each gear and engine map. The information from the API doesn’t have this information so thats why this was created. The XSD is the schema for the xml file and also contains the class that the xml data is read into by the application.

ArduinoInterfaces Project

Constants.cs contains all the constants used by the arduino interface and the iRduino application. I tried to use constants instead of just using numbers like 8 which have less meaning then NumberOfLEDsonTM1638.

ArduinoLink.cs contains:
The “font” used to display text on the TM displays.
Code to manage the Serial Port.
Code to list all the available COM ports.
Code to receive messages from the Arduino.
Code to send messages to the Arduino.

The send and receive methods work on their own threads so that the application remains responsive. The serial messages are a custom message type. Each message type has a messageID so that the application/arduino knows what to do with the information. This allows the data that needs to be sent to the arduino to be sent in many different packets. Each message type can thus to sent at different refresh rates. For example, we can send the LEDs information 30 times a second and only send the display text 15 times a second. This allows the serial port to be more flexible and require less data to be sent.

Message Format:
StartByte
MessageID
...MessageData...
CheckSum (of messageID and MessageData together)
MessageDataLength
EndByte

You can look in the constants for the messageIDs already used.

iRduino Project

ArduinoTemplates folder contains a T4 text template for generating the Arduino Sketch. There is a sample .ino file there as well but it is probably out of date now.

The classes folder contains all the classes for the application. I'll explain each class later.

The MahApps.Metro folder contains the dlls for the window styling that looks like windows 8 applications.

The resources folder contains all the images,etc.. used by the application.

The windows folder contains all the user interface stuff like windows and pages. The important windows are MainWindow which is the starting point of the application and the OptionsWindow. There is quite a lot of logic contained within the cs files for those windows. I'll go over those later.

The pages folder contains all the pages that used in the OptionsWindow. Everything that is displayed in the main part of the options window is a page that is load from here.

MainWindow
MainWindow is the starting point of the application. The variables for the DisplayManager, ArduinoConnection, and the SDK wrapper are stored as fields in this window. A lot of the intialisation of variables is done in the MainWindowLoaded method. The SDK wrapper is created in this method. The SDK wrapper exposes some events which is used by the applciation. Everytime the telemetry or SessionInfo is updated then it fires an event which is used to update everything in the app. There is also some code for loading all the configurations in folder in this windows class file.

OptionsWindow
This is a complex window. It has a pageframe which is used to display an appropriate options page. Everytime that a page is loaded then it changes the frames data context so that each page can access the data it needs. Some button clicks and selection changes in the pages are not handled by the page itself but are bubbled up to the OptionsWindow which handles them; these are the routedcheck, routedbutton, and routedselection functions. The tree view functions are rather complex too, the buildtreeviewnodes function builds the tree list (it is a recursive function that calls itself). Changing pages wasn't so simple because I was getting problems with it. It needs to change the page to a blank page first and then wait until it has finished loading that blank page before trying to load the next page that it wants to display. This was done to get around some problems. There are also functions for loading, saving, removing, etc of the configuration files.

Classes
AdvancedOptions.cs
This is a class for functions that are used by the Advanced Options page of the options window. Mainly the functions are used for converting the values entered in the UI back into

usable values for the application. eg. changing a string of “19200” to an int of 19200 with the necessary checks. The possible refresh rates and serial speeds are contained here.

ArduinoMessageReceiving.cs
Code that takes a message received from the serial port and sends the message data to an appropriate function according to the messageID. Currently only has code for receiving button presses on a Tm1638. Message[0] is the messageID and the rest of Message is the MessageData

ArduinoMessageSending.cs
Code that generates the MessageData for sending along the serial port to the Arduino. Only creates the message data. Doesn’t have the startbyte, endbyte, messageID, checksum and the rest that are part of the final message to the arduino.

ArduinoSketch.cs
Code that creates the arduino sketch file and the folder it sits in. Uses the T4 template code to generate the sketch contents.

ButtonFunctions.cs
Contains almost everything that is required to make button functions. The enum contains all the different types of button functions available. There are some dictionaries that contain the display names used in the UI.

Configuration.cs
This is the class that stores the configuration information. It has code for saving and loading the configuration from file. The configuration file on the hard-drive is basically a text file (kinda like a csv file as well).

ConfigurationOptions.cs
This is class that contains the configuration for the options window. It is different from the main configuration class because it needs to be data-bindable for the UI elements. Everyhting is a property so that it can be data-bound. For it to be data-boundalbe the properties sometimes need to be in different data types. For example, preferredComPort is an integer which represents the selectedIndex of the combobox (i use comboboxes and checkboxs as much as possible so that the user can’t enter strange values), however the main configuration stores this as a string such as “COM4”. Thus it needs to be converted from COM4 to 3 when loading from a configuration and converted from 4 to COM5 when saved back from a ConfigurationOption to a configuration. There are functions for converting (loading and saving) to and from a configuration to a configurationOptions class. I’m not so good at data-binding, so I’m sure that there is a much easier way to handle all of this. So to create a new configuration option such as “Auto Start connection”, it would need to be added to the appropriate options page, have a boolean property in configurationOptions.cs and a bool property in configuration.cs, be included in the converting functions (within configuationOptions.cs), and be included in the loading and saving functions of configuration.cs and be data-bound in the page code. Thats just for adding the UI stuff for the feature. Actually implementing something like an Auto-start would be rather simple in the MainWindow.

ControllerDevice.cs
Code for handling a usb controller

DCVariables.cs

Code for reading and displaying Dashboard Control variables from the iRacing API

Dictionarys.cs
Class for storing dictionarys used by the applciation. The dictionarys are mainly used for converting between string and enums.

DisplayManager.cs
Most important Class of the whole application probably. Handles all the values from the API and contains all the code to do all the useful stuff. Have a good detailed look through this file. It basically takes the information from the iracing APi and according to the configuration it creates all the messages that get sent to arduino. Things like determining which leds to light up and what to display on the screens.

DisplayVariables.cs
All the variables that can be displayed on the screens.

ErrorReporting.cs
Creates the error window.

LapDisplay.cs
Lap time display functions. Lap time display types

LEDs.cs
Shift light functions and shift light styles.

OptionPages.cs
Enum of option page types, and a function for getting the current version of the application.

ShiftLightData.cs
Function to load ShiftLightData.xml