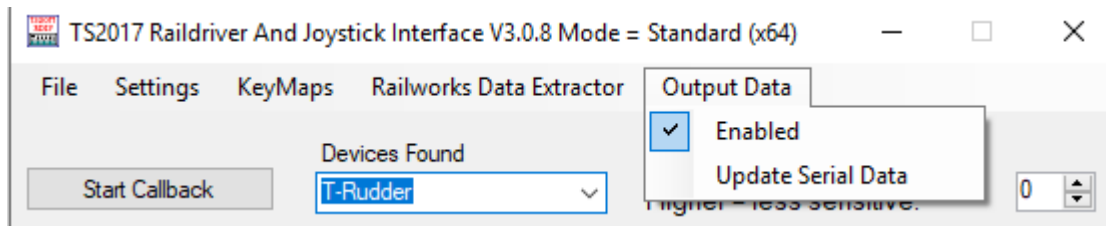


Output Data To SerialPorts.

As off version 3.0.8 you can now output the majority of the data that is displayed in the overlay in Advanced mode, to any serial port(s) attached to your PC. This can be done in either mode.

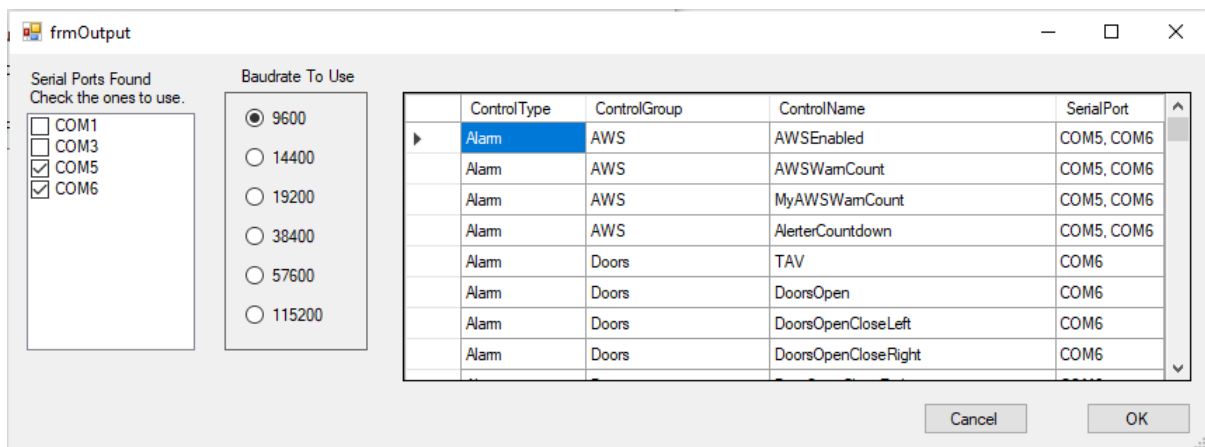
To enable this, the following has been added to the program.

1.



If the program detects any serial ports attached then it will add the “Output Data” menu option shown above. You then need to check the box next to “Enabled” to show the “Update Serial Data” menu option, which is used to assign various controls to each serial port (see 2 below). If you don’t check the “Enabled” menu option then nothing will be sent to the serial ports.

2. On clicking the “Update Serial Data” option you will be presented with the following

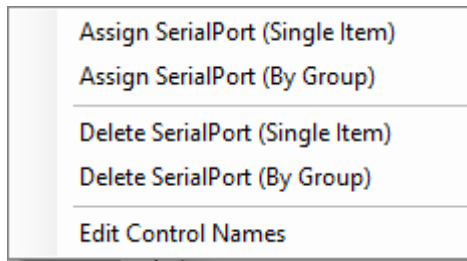


The first box on the left displays a list of all the serial ports found on your PC. You need to put a check in the ones you wish to send data to.

The second box is for selecting a baud rate to use for the serial ports. I have found the communications between the PC and Arduino to be fine using the default 9600 but can show errors on a LCD display as you increase the baud rate.

The final box shows the list of controls taken from the ControlNames.txt file that is used throughout the program. You have the ControlType which is Alarm/Lever/Gauge etc., then the ControlGroup which is Reverser/Throttle/TrainBrake/Doors etc., Next you have ControlName which is the actual name of the control in the train you are driving and finally you have the serial port(s) that the data will be sent to.

If you right click on any of the rows you will be presented with the following menu

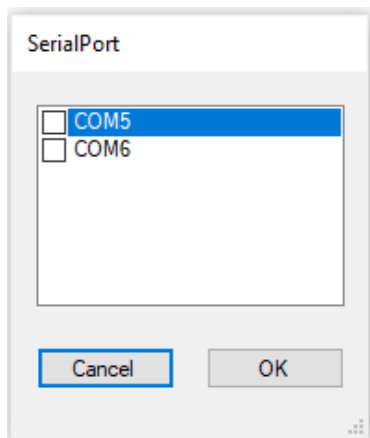


Assign SerialPort (Single Item) will allow you to select a serial port(s) for that row only.

Assign SerialPort (By Group) will allow you to select a serial port(s) for all the rows that have the same ControlGroup as the row you clicked on i.e. as in all the AWS assigned to COM5 and COM6 in picture 2.

The two delete serialport options do the same but will delete the assigned serial port(s).

When clicking either of the above Assign options you will be presented with the following window



The window will only list the serial ports you have ticked at 2 above, simply select the port(s) required and click OK. All the settings and assignments are save ready to be reloaded each time the program is run.

The final option on the menu above is “Edit Control Names”, this works the same as the “Settings\Edit Control Names” menu option. You must be careful to keep the format the same or the program will fail. The format is:-

ControlType=ControlGroup=ControlName=DisplayName=DisplayUnits(MPH/KPH, PSI/Bar, Gallons/Litres etc.). The DisplayName and DisplayUnits are only used in the overlay which is displayed when you are running the program in Advanced Mode. These can be left off if using the Standard Mode. Examples are

Lever=Reverser=VirtualReverser

Gauge=BoilerPressure=BoilerPressureGaugePSI=Boiler Pressure=PSI

Alarm=Sifa=SiFaWarning=Sifa.

Once you have finished updating the data, click OK on the frmOutput screen at 2 above and when you click the “Start Callback” button, the selected data will be sent to the relevant serial ports.

The data sent is in the format <ControlGroup:ControlName:value>.

E.G. <Reverser:VirtualReverser:0.5> we use < and > to indicate the start and end of the data and : to split the info.

Next Speed Limit/Previous Speed Limit Added in V3.1.2

As of V3.1.2 if you are using Advanced mode and put the reverser into reverse, then the data for the NextSpeedLimitBackSpeed and NextSpeedLimitBackDistance will be sent to the serial port instead of NextSpeedLimitSpeed and NextSpeedLimitDistance but with a negative number to show that it is the speed limit behind being displayed. If you switch to the rear cab then the speed limits should swap so that what was the speed limit behind, now becomes the speed limit in front. Unfortunately this does not work for all loco's so I have included a macro called “MACRO_TOGGLE_SPEEDLIMIT” which will switch forward/back speed limits on each call. To use it simply edit your button map, select the “Add New Command “ from the right click menu option, give the Control Name box a name (I used “Switch Speed Limits”) and select the “MACRO_TOGGLE_SPEEDLIMIT” from the Key box, it is the last item in the list. This does the same as pressing Shift+Alt+D when the overlay is displayed.

The data sent to the serial port will be as follows. For the speed limit ahead,
SpeedLimit:NextSpeedLimitSpeed:50.00 and SpeedLimit:NextSpeedLimitDistance:50
And for the speed limit behind,
SpeedLimit:NextSpeedLimitSpeed:-50.00 and SpeedLimit:NextSpeedLimitDistance:-50

I have supplied an Arduino script file called “ArduinoFromTS2019.ino” in the folder “ArduinoFromTS2019” which demonstrates how to use this data to light led's and display the Speed, Gears, Reverser and Doors on a 20x4 I2c lcd. I have placed a copy of the code at the end of this document starting on page 4. Here is a picture of the program in action



You may find that with some of the data you are not getting the results you expect. An example is the AWS. If you want to have a light show when the AWS alarm sounds then use the Alarm=AWS group but if you want a light to function the same as the sunflower then you need to use the Warning=Sunflower=AWS I have added at the bottom of the ControlNames.txt file.

I have also added two more warnings to the file

Warning=DRA=DRA

Warning=DRA=DRAButton

This will allow you to have a led light when the DRA is on.

If you find that a function is not showing when you assign the serial ports then do the following.

1. Run TS and the program and drive the loco.
2. Pause TS and then Alt+Tab to my program and scroll down the right hand panel to see if you can see the name of the control.
3. If you can then go to 5.
4. If you don't know the name of the control you are looking for then the easiest way to find it is to select all the text in the panel and copy it to a text editor like Notepad++, then unpause TS, move the control, pause again and copy the updated text in the right panel to a second window in Notepad++. Then compare the two files, noting which controls have changed, this should give you the control name.
5. Now use the Settings\Edit Control Names option and add the control to the bottom of the list. The ControlType and ControlGroup can be anything you like, I have only used Warnings because it places it at the bottom of the list and makes it easier to find and used Sunflower as it makes it obvious what the AWS control is controlling. Make sure you keep the format correct though.
6. Now select the Output Data\Update Serial Data menu option and assign the serial port you wish to use. You can sort the columns by clicking on the column names.

Here is the code for the Arduino.

```
/*
 * This is a test sketch for a Arduino to allow the data received via its serialport
 * from the "TS2017 Raildriver And Joystick Interface" program to be used to control
 * led's and LCD displays. The data is in the format
 * "<ControlGroup:ControlName:value>".
 * E.G. <Reverser:VirtualReverser:0.5>
 * we use < and > to indicate the start and end of the data and : to split the info.
 */
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
const byte AwsDRA_Pin = 13;
const byte Doors_Pin = 2;
const byte Dsd_Pin = 3;
const byte Emergency_Pin = 4;
const byte Engine_Pin = 5;
const byte Handbrake_Pin = 6;
const byte Sifa_Pin = 7;

//Here we add the details regarding the lcd screen width/height
const byte lcdWidth = 20;
const byte lcdRows = 4;

//Buffer to hold data from serialport
//data[0] = controlgroup
//data[1] = controlname
//data[2] = value
char data[3][80];

// byte data to hold the position of the current char in the data array.
byte cnt = 0;
```

```

byte pos = 0;

//Marker set when start of data received
boolean started = false;

//Marker set when all data received
boolean finished = false;

//Set I2C LCD Address
byte Lcd_Address = 0x27;

//Initialise the lcd.
LiquidCrystal_I2C lcd(Lcd_Address, 20, 4);

char ButtonMap[80];
char LocoName[80];

void setup() {
  // Set the pins
  pinMode(AwsDRA_Pin, OUTPUT);
  pinMode(Doors_Pin, OUTPUT);
  pinMode(Dsd_Pin, OUTPUT);
  pinMode(Emergency_Pin, OUTPUT);
  pinMode(Engine_Pin, OUTPUT);
  pinMode(Handbrake_Pin, OUTPUT);
  pinMode(Sifa_Pin, OUTPUT);
  Serial.begin(9600);

  //Setup lcd
  lcd.begin();
  lcd.backlight();
  lcd.clear();

  //Display a message to the user to show the code is working
  // and waiting for data.
  lcd.setCursor(0,0);
  lcd.print("Waiting for TS2019");
}

void loop() {
  // Check we have something in the serial buffer
  while(Serial.available() > 0) {
    char nextChar = Serial.read();
    if(nextChar == '<') { //Check for start of command marker
      started = true;
      finished = false;
      pos = 0;
      cnt = 0;

      //Reset data to nothing

```

```

    data[0][0] = '\0';
    data[1][0] = '\0';
    data[2][0] = '\0';
}
else if(nextChar == '>') { //End of command marker
    finished = true;
    break; //We have our data so exit the loop
}
else if(nextChar == ':') { //End of current string
    pos++;
    cnt = 0;
}
else {
    data[pos][cnt] = nextChar; //Store the incoming data one char at a time
    cnt++;
    data[pos][cnt] = '\0'; //Null terminate the data array;
}
} //End of While loop

if(started && finished) { //All data for the current command received.
    UpdateHardware(); //Activate switches/displays etc
}
}

void UpdateHardware() {
    //Check if loco has changed, if so clear the display.
    if(!strcmp(data[0], "locochanged")){
        // Clear the lcd screen
        lcd.clear();
        //Turn all led's off
        digitalWrite(AwsDRA_Pin, LOW);
        digitalWrite(Doors_Pin, LOW);
        digitalWrite(Dsd_Pin, LOW);
        digitalWrite(Emergency_Pin, LOW);
        digitalWrite(Engine_Pin, LOW);
        digitalWrite(Handbrake_Pin, LOW);
        digitalWrite(Sifa_Pin, LOW);
        strcpy(ButtonMap, data[1]);
        strcpy(LocoName, data[2]);
        return;
    }

    //Convert value to an int.
    int value = atoi(data[2]);

    //Convert value to a float.
    float fvalue = atof(data[2]);

    //Used to hold how many spaces required at end of a line for the lcd
    byte padding = 0;

```

```

//Temporary variable used when formatting output string for the display
char tmp[30];

//Although we have both sunflower and DRA set to the same pin, because in the program
//I have them assigned to different serialports, only the command assigned to this
//Arduinos port will be activated, this allows the same code to be used on multiple Arduinos.
if( !strcmp(data[0],"Sunflower") || !strcmp(data[0], "DRA")) {
    if(value == 1) {
        digitalWrite(AwsDRA_Pin, HIGH);
    } else if(value == 0) {
        digitalWrite(AwsDRA_Pin, LOW);
    }
}
//Here we check to see if the controlgroup is "Speed", if it is, display it
else if( !strcmp(data[0], "Speed")){
    lcd.setCursor(0,0);
    //The sprintf() function in the Arduino does not allow the use of float variables
    // so we need to use the dtostrf() function to convert the value to a string.
    char newVal[7];
    dtostrf(fvalue, 4, 1, newVal);
    sprintf(tmp,"%s = %s", data[0], newVal);
    padding = lcdWidth - strlen(tmp);
    lcd.print(tmp);
    for(int i = 0; i < padding; i++) {
        lcd.print(" ");
    }
}
//Here we move the cursor to line 1 and display the controlgroup "Gears"
else if( !strcmp(data[0], "Gears")) {
    lcd.setCursor(0,1);
    sprintf(tmp, "%s = %d", data[0], value);
    padding = lcdWidth - strlen(tmp);
    lcd.print(tmp);
    for(int i = 0; i < padding; i++) {
        lcd.print(" ");
    }
}
//Here we move the cursor to line 2 and display the controlgroup "Reverser"
//We use the float value and multiply it by 100 to get the % value. We then
//add the % sign to the end.
else if( !strcmp(data[0], "Reverser")) {
    lcd.setCursor(0,2);
    int i = fvalue * 100;
    sprintf(tmp, "%s = %d%%", data[0], i);
    padding = lcdWidth - strlen(tmp);
    lcd.print(tmp);
    for(int i = 0; i < padding; i++) {
        lcd.print(" ");
    }
}
//Here we display text to show if the doors are open/closed

```

```

else if( !strcmp(data[0], "Doors")) {
    lcd.setCursor(0,3);
    if(value >= 1)
        sprintf(tmp, "%s Open", data[0]);
    else if(value == 0)
        sprintf(tmp, "%s Closed", data[0]);
    padding = lcdWidth - strlen(tmp);
    lcd.print(tmp);
    for(int i = 0; i < padding; i++) {
        lcd.print(" ");
    }
}
}
}

```

28/01/20

The program now outputs the name of the ButtonMap without the .xml extension and the LocoName, whenever the loco changes. To enable this I have added the code

```
char ButtonMap[80];
```

```
char LocoName[80];
```

in the main declaration section and then in the 'if(!strcmp(data[0], "locochanged"))' section of the UpdateHardware() function I added

```
strcpy(ButtonMap, data[1]);
```

```
strcpy(LocoName, data[2]);
```

When the program detects a new loco it sends "<locochanged:ButtonMap:LocoName> "so Data[0] = "locochanged", data[1] = name of the current ButtonMap, data[2] = name of current loco.

16/04/20

You can now output the same data to more than one serial port.