RoboDAQ7

Masters of Engineering Report

By John Buzzi

Email: jlb269@cornell.edu

Cornell University

May 17, 2010

Abstract

Learning from and improving on our past mistakes and accomplishments is only possible if we understand what we have done. This statement is particularly true in robotics. It may be possible to create a working robot with the guess and check method, but in order to systematically debug and develop a robot more sophisticated feedback is essential. In order to have real-time interactions and feedback with the Cornell Ranger walking robot the RoboDAQ7 program was developed. This program was designed from the ground up to work hand in hand with the Ranger to facilitate progress. The RoboDAQ7 program is an entirely redesigned version of a previous data acquisition system, in fact the entire user interface was deleted and reprogrammed to be optimized for the new data handling schemes on the Ranger robot. This report aims to explain what the RoboDAQ7 system is, why it is needed, how to use it, and how to modify it.

Purpose and Need

The purpose of the RoboDAQ7 data acquisition software is to allow a simple way to perform complex interactions with a functioning robotic system. In our case the system of interest is the Cornell Ranger walking robot. The RoboDAQ7 system performs three main functions:

1. Stores data being sent from the robot for post-testing analysis by other programs
2. Allows the user to view incoming data in real time.
3. Allows the user to modify parameters on the robot during operation without having to reload the robot code each time a change is needed.

All three of these tasks were designed for the purpose of allowing for the debugging and fine tuning of the robotic system. This ultimately aims to enable or expedite the creation a functioning robot.

Having a highly complex electro-mechanical system like the Ranger robot it is not always clear what is going wrong. Especially when trying to accomplish tasks such as walking many continuous miles, it is especially important to be able to detect and correct errors in a timely fashion. Without a means to do this it becomes more and more difficult to develop a working system as the level of complexity increases. It is safe to say that by having a real time 2-way interaction with the robot the chances of success become much greater.

The RoboDAQ7 System

In order to create a system that achieves the above mentioned 3 primary tasks the National Instruments LabVIEW programming language was chosen as the clearest way of doing this. LabVIEW is primarily designed for data acquisition purposes and the development of user interfaces for a variety of machines. LabVIEW also lends itself well to running parallel processes which is useful for this type of system.

In order to develop a system like this there are a lot of pieces that have to work together simultaneously. The way our system was organized is that one program RoboDAQ7.vi is called to initialize 3 other parallel programs. RoboParser7.vi is the program responsible for reading the incoming data and storing it to a data file. RoboControl7.vi is a program which sends values from LabVIEW back to the robot such as parameters. RoboDisplay7.vi is the primary user interface and will be the main focus of this report. Attached to this document are the User's Manual, which explains how to use the RoboDAQ program, and Design Manual, which explains how the program works in case modifications need to be made.

RoboDAQ7 Users Manual

By John Buzzi

Email: jlb269@cornell.edu

May 17, 2010

Contents

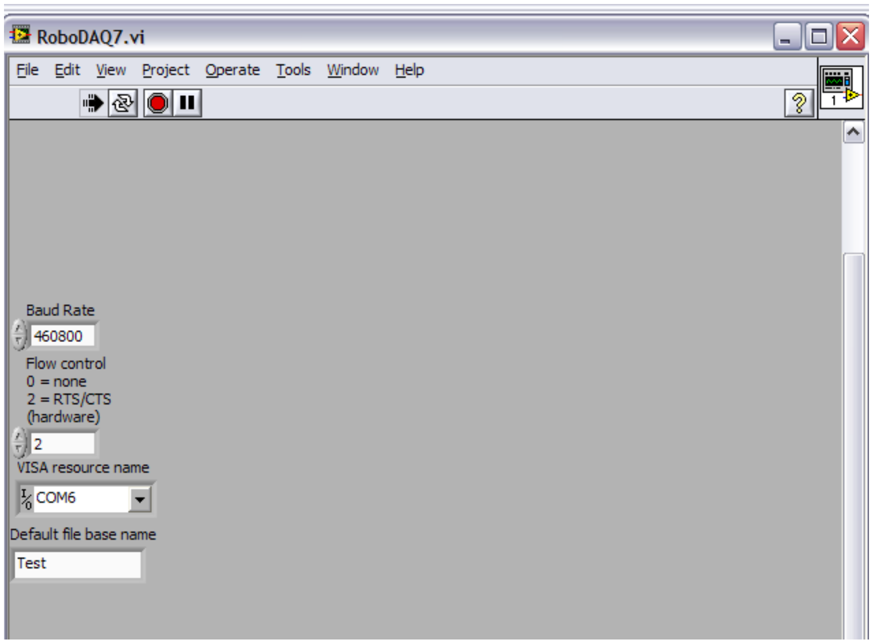
Introduction

The purpose of this manual to act as a reference for people interested in using the RoboDAQ7 program. This is not intended to be a guide for those wishing to make modifications to the code; however it may prove useful to understand in that regard as well.

Getting Started

In order to begin using the LabVIEW Robot Data Acquisition System Version 7 (RoboDAQ7) you must first navigate to the correct folder in the lab's current SVN system. As of 5/18/10 the program is located in the directory <…\SVN\Trunk\Data_Display\LabView_DAQ_Display\ROBO_DAQ_7\RoboDAQ7.vi>. This RoboDAQ7.vi file is the initialization file for the rest of the program and must be run first. Double click the file to start LabVIEW and you will first see the screen in Figure-1.As long as you are using the Bluetooth module to acquire data leave all input fields at their default values except for "VISA resource name" which you must set to the COM port value that corresponds to the Bluetooth adapter. You may choose the COM port by selecting the drop down menu. You may also choose to rename the default data file base name which will automatically be "Test" if not otherwise specified. Note that this file name will also have the date and time stamp of when it was generated appended to it automatically.

Upon completing the hardware setup, click the ⇨ button to run the program.



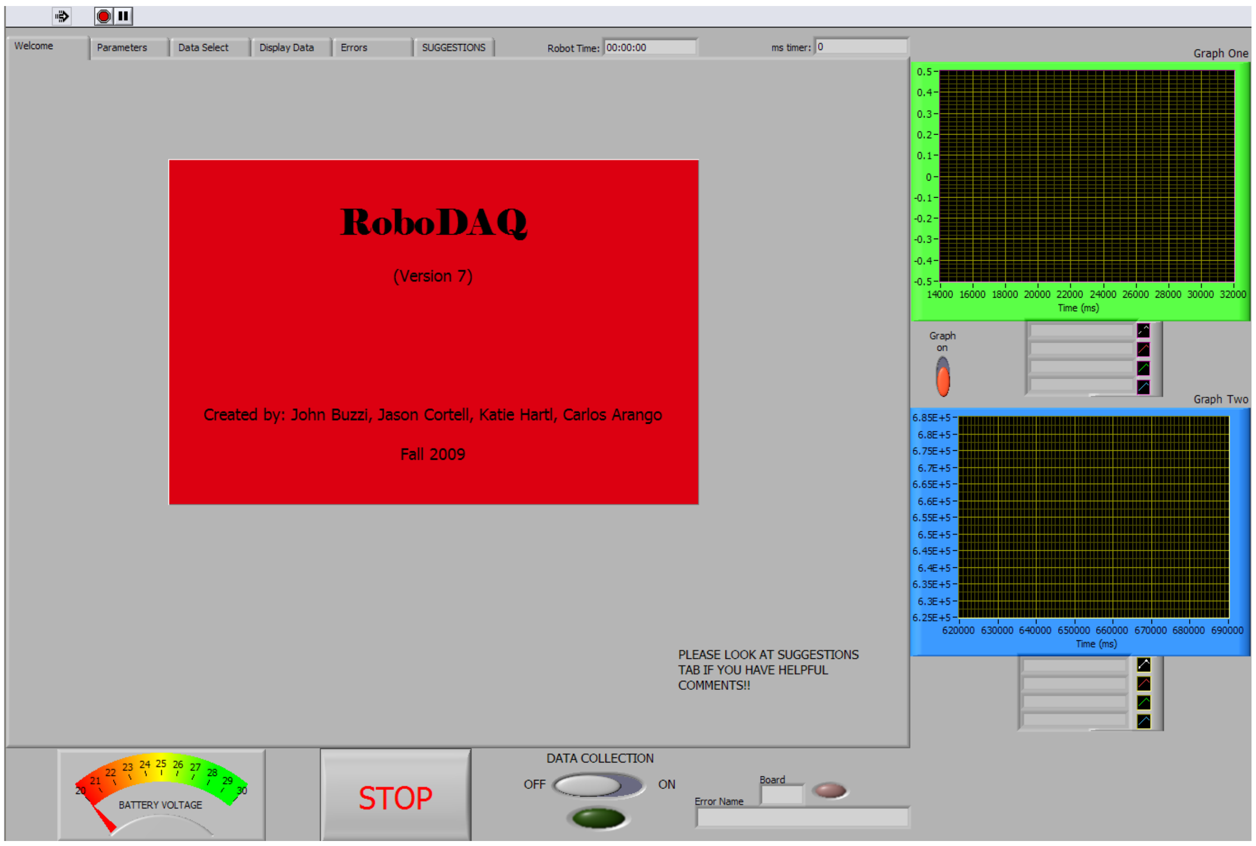
Baud Rate
460800
Flow control
0 = none
2 = RTS/CTS
(hardware)
2
VISA resource name
COM6
Default file base name
Test

Figure 1: RoboDAQ7 initialization screen

After you begin running the program you will be prompted for 4 file directories. The first of which is the directory where any data files will be saved to. Next you will be prompted to specify the CAN ID file that will be read into LabVIEW with a full list of the parameters and data channels that are being used in the robot code. The default file path for this is <\SVN\Trunk\Ranger\Control\Ranger_Configuration\can_id.h> and it is recommended that you use this file unless you know for certain that a different CAN ID file is loaded on the robot. Third you will be prompted to specify the Board ID file that contains the assigned board identification numbers corresponding to each board on the robot. Once again the default file path for this is given by <\SVN\Trunk\Ranger\Control\Ranger_Configuration\board_id.h> and it is recommended that you use this file unless you know otherwise. Finally you will specify the Error ID file in the same manor. The default location is <\SVN\Trunk\Ranger\Control\Ranger_Configuration\board_id.h>.

Welcome Tab

If all files were successfully selected, the RoboDisplay7.vi welcome screen will open. This page simply displays the program version and acknowledges the people who worked on it. The program, as you will notice in Figure-2, is designed around a "tabbed" user interface in order to maximize usage of the given screen space. It is recommended that when using the program for the first time that you make your way from left to right across the Tabs in order to familiarize yourself with what the function of each tab is.



RoboDAQ

(Version 7)

Created by: John Buzzi, Jason Cortell, Katie Hartl, Carlos Arango

Fall 2009

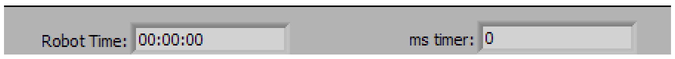
PLEASE LOOK AT SUGGESTIONS TAB IF YOU HAVE HELPFUL COMMENTS!!

Figure 2: Welcome Screen and overall view of RoboDisplay7 program

Non-Tabbed Front Panel Displays

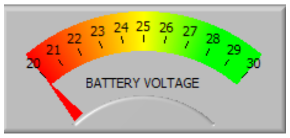
In addition to the Tab panel, there are other displays which are always visible that may be of interest to the user.

Robot Time Display



For starters the robot time in "Hour:Minute:Second" format as well as in millisecond format can be found along the top in the center. These timers show the most recent value of the ms timestamp that has been sent to LabVIEW from the robot. You can easily check to see if the robot is correctly sending data by checking if these fields are being updated continuously.

Battery Voltage Display

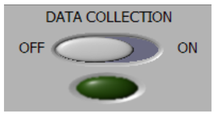


While not currently functioning (As of 5/18/10 due to a lack of accurate battery voltage data), the battery voltage display is designed to be a straightforward way of understanding the amount of stored electrical energy left in the robot. Ideally this would be a clear way to see if the battery voltage is significantly impacting performance of the robot.

Stop Button

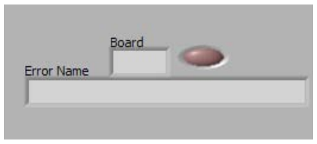
The **STOP** button is the easiest way to stop the execution of all LabVIEW RoboDAQ programs simultaneously.

Data Collection Toggle



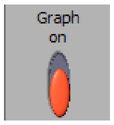
By default the RoboDAQ program does not automatically collect data upon initialization. In order to turn data collection on you must switch the **Data Collection** Toggle to the **ON** position. Upon pressing this toggle the green light will turn on indicating that data is being collected and a new test file will be generated. When the data collection is turned **OFF** the test file is closed and is not accessed by LabVIEW again. Note: The LabVIEW program will automatically turn data collection off if the robot is turned off and back on again (resetting the timestamp).

Quick Error Display



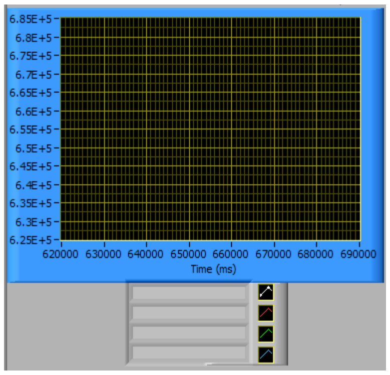
As a quick way of seeing what errors have occurred on the Robot during a given test run you may refer to the Quick Error Display. This indicator cycles through all current errors and displays the Error Name and Board Abbreviation for 2 seconds continuously.

Graph on/off Toggle



This toggle must be switched up to the **Graph On** position in order to enable graphing in this program. The toggle is set to off by default in order to minimize computer usage until graphing is needed.

Graphs

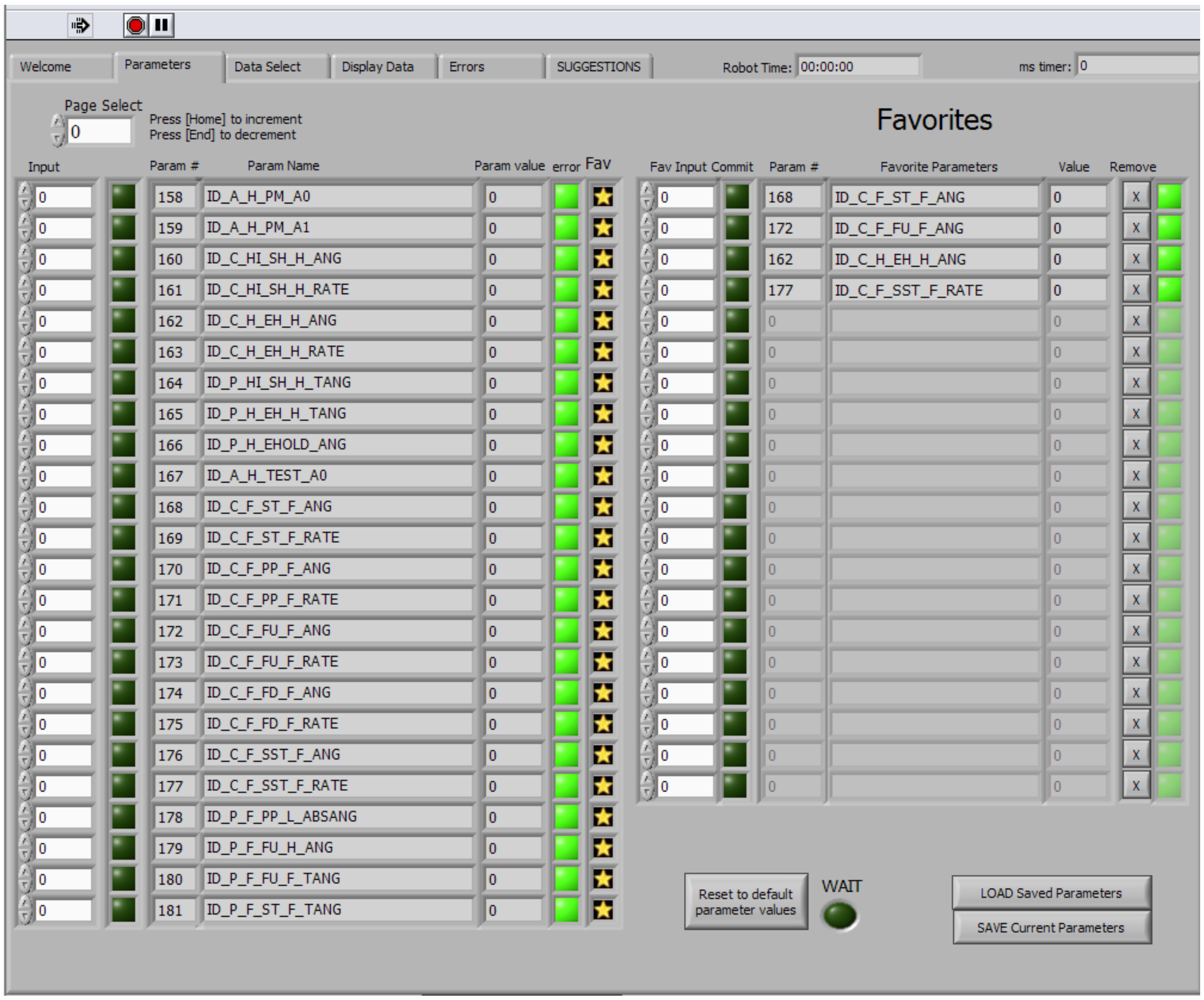


Located to the right of the screen are two individually controlled graphs. Each graph is capable of plotting up to 4 data channels at a time. The axes are auto-scaled by default but if you are want to look at a particular range you may right click the axis of interest and turn Auto-Scaling off. Once auto-scaling is off you may double click the scale and modify the range manually. The legend located below each graph indicates which variables are being plotted at any given time.

Parameters Tab

Purpose

The purpose of this tab is to allow the user to change variables on the robot, called parameters, from LabVIEW in real time. This can be particularly useful for tuning gains or setting angles on the robot without having to modify and reload the code to the Main Brain. The Parameters Tab can be seen in Figure-3.



Input		Param #	Param Name	Param value	error	Fav
0		158	ID_A_H_PM_A0	0	🟩	★
0		159	ID_A_H_PM_A1	0	🟩	★
0		160	ID_C_HI_SH_H_ANG	0	🟩	★
0		161	ID_C_HI_SH_H_RATE	0	🟩	★
0		162	ID_C_H_EH_H_ANG	0	🟩	★
0		163	ID_C_H_EH_H_RATE	0	🟩	★
0		164	ID_P_HI_SH_H_TANG	0	🟩	★
0		165	ID_P_H_EH_H_TANG	0	🟩	★
0		166	ID_P_H_EHOLD_ANG	0	🟩	★
0		167	ID_A_H_TEST_A0	0	🟩	★
0		168	ID_C_F_ST_F_ANG	0	🟩	★
0		169	ID_C_F_ST_F_RATE	0	🟩	★
0		170	ID_C_F_PP_F_ANG	0	🟩	★
0		171	ID_C_F_PP_F_RATE	0	🟩	★
0		172	ID_C_F_FU_F_ANG	0	🟩	★
0		173	ID_C_F_FU_F_RATE	0	🟩	★
0		174	ID_C_F_FD_F_ANG	0	🟩	★
0		175	ID_C_F_FD_F_RATE	0	🟩	★
0		176	ID_C_F_SST_F_ANG	0	🟩	★
0		177	ID_C_F_SST_F_RATE	0	🟩	★
0		178	ID_P_F_PP_L_ABSANG	0	🟩	★
0		179	ID_P_F_FU_H_ANG	0	🟩	★
0		180	ID_P_F_FU_F_TANG	0	🟩	★
0		181	ID_P_F_ST_F_TANG	0	🟩	★

Fav Input	Commit	Param #	Favorite Parameters	Value	Remove	
0		168	ID_C_F_ST_F_ANG	0	X	🟩
0		172	ID_C_F_FU_F_ANG	0	X	🟩
0		162	ID_C_H_EH_H_ANG	0	X	🟩
0		177	ID_C_F_SST_F_RATE	0	X	🟩
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	
0		0		0	X	

Figure 3: Parameters Tab

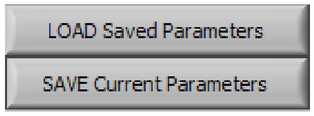
Changing a Parameter

The user can scroll through the list of parameters on the left hand side by toggling the value of **Page Select** and searching for the Parameter Name or ID Number that is desired. Once the parameter of interest is found the current value can be viewed in the "Param Value" column. This Param Value column displays only values that are read in from the robot so whatever value you see there is the current value and not just a desired value. In order to change this value you must enter the new desired value in the appropriate Input field and press the dark green commit button immediately to the right of the Input field. After a moments delay the new value should appear in the Param Value column. The error light should indicate whether or not the Input value matches the parameter value that the robot is sending out.

Favorites

The right hand side of the page contains the slots for 20 “favorite” parameters. This feature is for parameters that the user plans on using multiple times and instead of having to search through many pages of parameters they will be ready for quick modification at all times. In order to make a parameter a favorite simply press the button next to the desired parameter and a copy of it will be moved to the favorites. In order to remove a Favorite Parameter press the button next to the favorite that you want to remove. Changing values of a Favorite Parameter is done in the exact same way as the

Load/Save Parameters

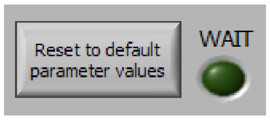


Current robot parameters are saved in a tab delimited text file as 5 rows of information organized as follows:

Row 1: Parameter Names
Row 2: Parameter ID Numbers
Row 3: Parameter Values
Row 4: Favorite Parameter Names
Row 5: Favorite Parameter ID Numbers

Note: When loaded the parameter names are ignored and loaded in by parameter ID number. If you have for any reason changed the number of any parameter a saved file is no longer compatible.

Reset to Default Parameters



This button serves to reset the robot parameters to the values that are assigned in the comment section of the can_id.h file. The main purpose of this is so that once you find a good base set of parameters that work, you can hard code them into the can_id.h file and load them quickly and easily each time instead of each person making their own custom parameter files for everything.

Hints and Tips

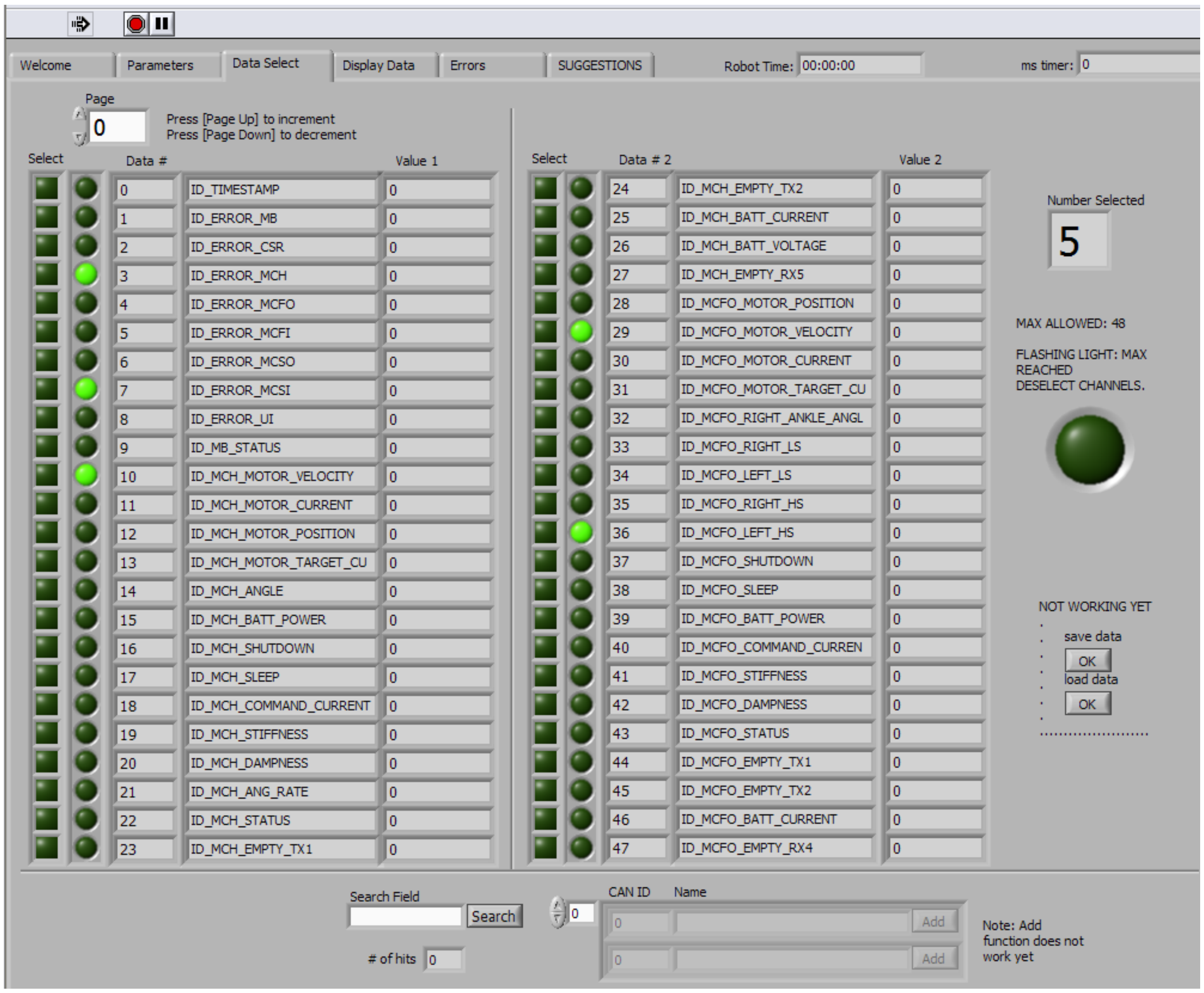
- Press the [Home] key to increment to the next parameter page
- Press the [End} key to decrement to the previous parameter page

- The red error light only means the desired input field does not numerically match the param value field. If you can verify that the value in the param value field is the value that you desire then you can ignore the red light if it goes on mistakenly.

Data Select Tab

Purpose

The purpose of this tab is to allow the user to select which data channels are of interest to them. In order for a data channel to be plotted or saved at a high transfer rate it must be selected in this tab. Any data channel that is not selected will be periodically sent out as slow data. The Data Select tab is shown below in Figure-4



Robot Time: 00:00:00 ms timer: 0

Page 0

Press [Page Up] to increment
Press [Page Down] to decrement

Select	Data #		Value 1	Select	Data # 2		Value 2
	0	ID_TIMESTAMP	0		24	ID_MCH_EMPTY_TX2	0
	1	ID_ERROR_MB	0		25	ID_MCH_BATT_CURRENT	0
	2	ID_ERROR_CSR	0		26	ID_MCH_BATT_VOLTAGE	0
●	3	ID_ERROR_MCH	0		27	ID_MCH_EMPTY_RX5	0
	4	ID_ERROR_MCFO	0		28	ID_MCFO_MOTOR_POSITION	0
	5	ID_ERROR_MCFI	0	●	29	ID_MCFO_MOTOR_VELOCITY	0
	6	ID_ERROR_MCSO	0		30	ID_MCFO_MOTOR_CURRENT	0
●	7	ID_ERROR_MCSI	0		31	ID_MCFO_MOTOR_TARGET_CU	0
	8	ID_ERROR_UI	0		32	ID_MCFO_RIGHT_ANKLE_ANGL	0
	9	ID_MB_STATUS	0		33	ID_MCFO_RIGHT_LS	0
●	10	ID_MCH_MOTOR_VELOCITY	0		34	ID_MCFO_LEFT_LS	0
	11	ID_MCH_MOTOR_CURRENT	0		35	ID_MCFO_RIGHT_HS	0
	12	ID_MCH_MOTOR_POSITION	0	●	36	ID_MCFO_LEFT_HS	0
	13	ID_MCH_MOTOR_TARGET_CU	0		37	ID_MCFO_SHUTDOWN	0
	14	ID_MCH_ANGLE	0		38	ID_MCFO_SLEEP	0
	15	ID_MCH_BATT_POWER	0		39	ID_MCFO_BATT_POWER	0
	16	ID_MCH_SHUTDOWN	0		40	ID_MCFO_COMMAND_CURREN	0
	17	ID_MCH_SLEEP	0		41	ID_MCFO_STIFFNESS	0
	18	ID_MCH_COMMAND_CURRENT	0		42	ID_MCFO_DAMPNESS	0
	19	ID_MCH_STIFFNESS	0		43	ID_MCFO_STATUS	0
	20	ID_MCH_DAMPNESS	0		44	ID_MCFO_EMPTY_TX1	0
	21	ID_MCH_ANG_RATE	0		45	ID_MCFO_EMPTY_TX2	0
	22	ID_MCH_STATUS	0		46	ID_MCFO_BATT_CURRENT	0
	23	ID_MCH_EMPTY_TX1	0		47	ID_MCFO_EMPTY_RX4	0

Number Selected 5

MAX ALLOWED: 48

FLASHING LIGHT: MAX REACHED DESELECT CHANNELS.

NOT WORKING YET
save data OK
load data OK

Search Field [Search]
# of hits 0

0

CAN ID	Name	
0		Add
0		Add

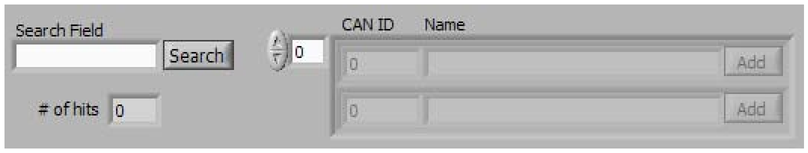
Note: Add function does not work yet

Figure 4: Data Select Tab

Selecting a Data Channel

In order to choose a data channel you would like to have more information on you must first locate the channel name or number by changing the **Page** field. Once you have found the channel of interest you then press the **Select** button 🟩 next to that channel. A circular green light ● will then light up next to the data channel indicating that it has been successfully selected. In order to De-select a data channel you must once again press the **Select** button and the green light should turn off.

Search for a Data Channel



It is possible that at some point there are going to be hundreds of available data channels to choose from and finding the one you are searching for could be difficult. In order to account for this a search function has been added to make things easier. In order to search for a data channel you must enter the search word in the **Search Field** box and then press the **Search** button. The function will then search all the names of the data channels for any strings that match your input. The function returns the total number of channels with that string as well as all of their full names and CAN ID numbers. The search is not case sensitive. Also note that the **ADD** buttons next to the data names are not functioning currently but would be extremely convenient as a future addition.

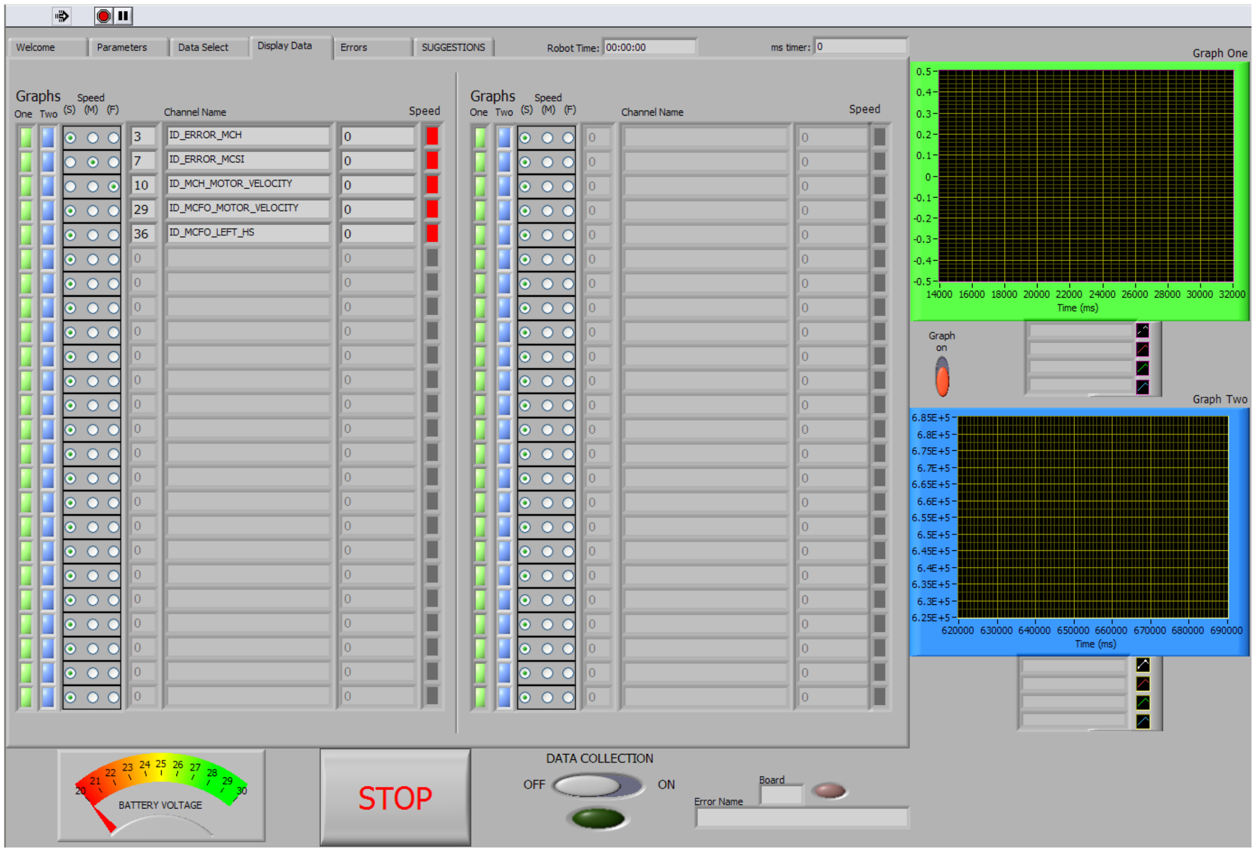
Hints and Tips

- Press the [Page Up] key to increment to the next Data Channel page
- Press the [Page Down] key to decrement to the previous Data Channel page
- Only 48 Data Channels may be selected at any given time, if you reached the limit the program will only allow you to de-select channels.

Data Display Tab

Purpose

The purpose of this tab is to select the data channels that will be plotted and set the speed at which data is sent from the robot. Speed in this case refers to how frequently data is sent out from the robot to LabVIEW. The Data Display Tab is shown below in Figure-5.



Robot Time: 00:00:00

ms timer: 0

Speed (S)	Speed (M)	Speed (F)		Channel Name	Speed
●	○	○	3	ID_ERROR_MCH	0
○	●	○	7	ID_ERROR_MCSI	0
○	○	●	10	ID_MCH_MOTOR_VELOCITY	0
●	○	○	29	ID_MCFO_MOTOR_VELOCITY	0
●	○	○	36	ID_MCFO_LEFT_HS	0

Figure 5: Data Display Tab

Graphing a Data Channel

The graphs on the right hand side of the screen can plot the previous 256 data points for each of up to 4 selected data channels. In order to enable graphing the **Graph On** toggle must be set to the up position. To choose which channels are plotted you must select either the blue or green buttons next to the data channel. Each button corresponds to the graph of the same color. For example if you would like to plot a variable on the blue graph you press the blue button to the left of the data channel you desire.

Changing the Data Speed

In order to set the rate at which a particular data channel is transmitting data to LabVIEW you must use the Speed radio button selector to the left of the data channel. From left to right the speeds are Slow, Medium, and Fast and are labeled (S), (M), and (F) to make this clear. While this toggle sets the data speed it does not indicate that the speed is successfully changed on the robot. The speed is indicated by the Speed color box 🟥 according to the following system:

Red – Slow speed data
Yellow – Medium speed data
Green – High speed data

It is always recommended that you trust these values over the radio button selection although they should usually match.

Errors Tab

Purpose

The purpose of this tab is to display a chronological list of the most recent errors to occur on the robot. The most recent errors are shown at the top of the list. The indicator fields are defined as follows:

- Board – Abbreviation of the board from which the error was reported as defined by board_id.h
- Error Name – Shorthand name for the error which occurred as defined by error_id.h
- Freq – The amount of times the error has occurred.
- Error Description – The description of the error given in the comments of the error_id.h file.
- Time – The most recent time in milliseconds at which the error was sent out

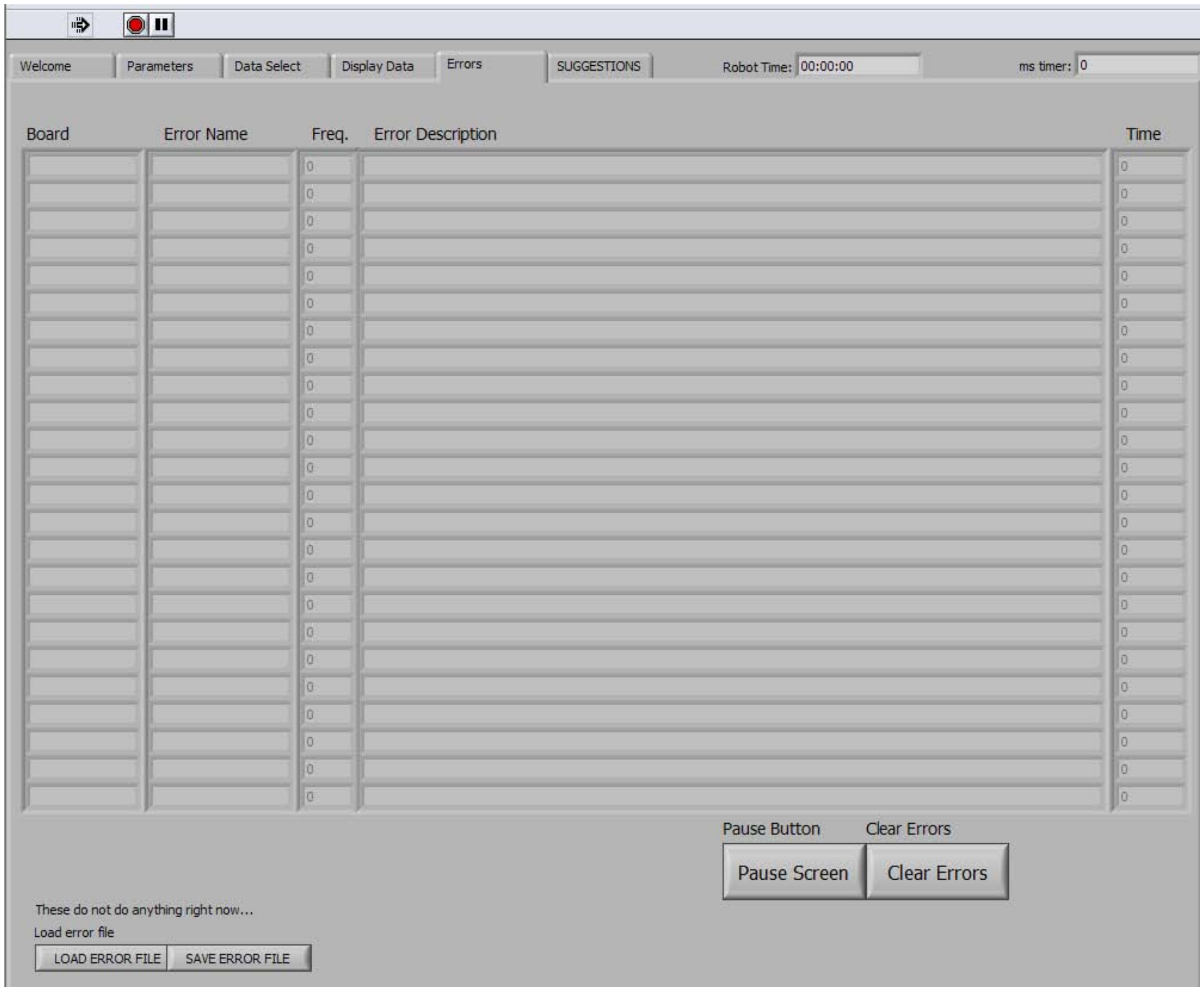
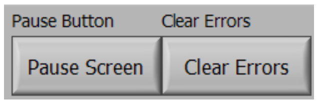


Figure 6: Errors Tab

Pause/Clear Errors



If for some reason there are too many errors occurring at once and the list becomes hard to read as it updates you may press the **Pause Screen** button. This button will stop updating this table until the button is pressed again. The **Clear Errors** button is used in case the robot is turned off and turned back on in order to reset the error list.

RoboDAQ7 Program Design Manual

By John L. Buzzi III

Email: jlb269@cornell.edu

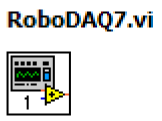
May 17, 2010

Contents

Introduction

This document describes how the Block Diagram section of the LabVIEW RoboDAQ system works. It is assumed that if you are reading this section that you have some minor experience with LabVIEW. Additionally I do not recommend that you attempt to edit the program unless you know fully how to use the program, I therefore suggest that you should read the RoboDAQ7 Users Manual before reading this unless you are fully familiar with how to use the program. This manual is breaks the RoboDAQ7 system down into its separate programs and explains key information about each section for the purposes of understanding and eventually editing the code. I have tried to make the users guide as visual as possible since LabVIEW is a visual programming language and it is extremely hard to describe things without being able to see them. You should note that this guide is still not entirely stand alone, the only way you can truly understand the program is to probe around in the code itself.

RoboDAQ7.vi



Purpose:

The purpose of this program is to initialize the entire LabVIEW Data Acquisition System. Multiple global variables are initialized here. Parser programs are called to read in the necessary CAN ID, Error ID and Board ID tables for the rest of the program to run correctly. This program also calls the three main parallel programs after initialization routines are completed. These programs are RoboDisplay7.vi RoboParser7.vi and RoboControl7.vi.

Location:

- RoboDAQ7 folder in SVN

Key Variables:

- GlobalErrorCAN_ID.vi
  - Error Board Names – Contains the board names corresponding to each error channel.
  - Error CAN_IDs – Contains the CAN ID numbers of each error channel.
- GlobalDataSize.vi – Contains the number of Data CAN ID channels.
- GlobalParamSize.vi – Contains the number of Parameter CAN ID channels.
- GlobalInitialParamVals.vi – Contains the initial values of each parameter read from the can_id.h file.
- GlobalLabVIEWStart.vi - Contains the value of the CAN ID just before the 48 LabVIEW speed channels.
- GlobalDataPath7.vi – Contains the path location of the save file that is being used.

- GlobalDataFileRefnum7.vi – Contains a LabVIEW path reference to the save file that has been opened so that it can be closed at a later time.

Inputs:

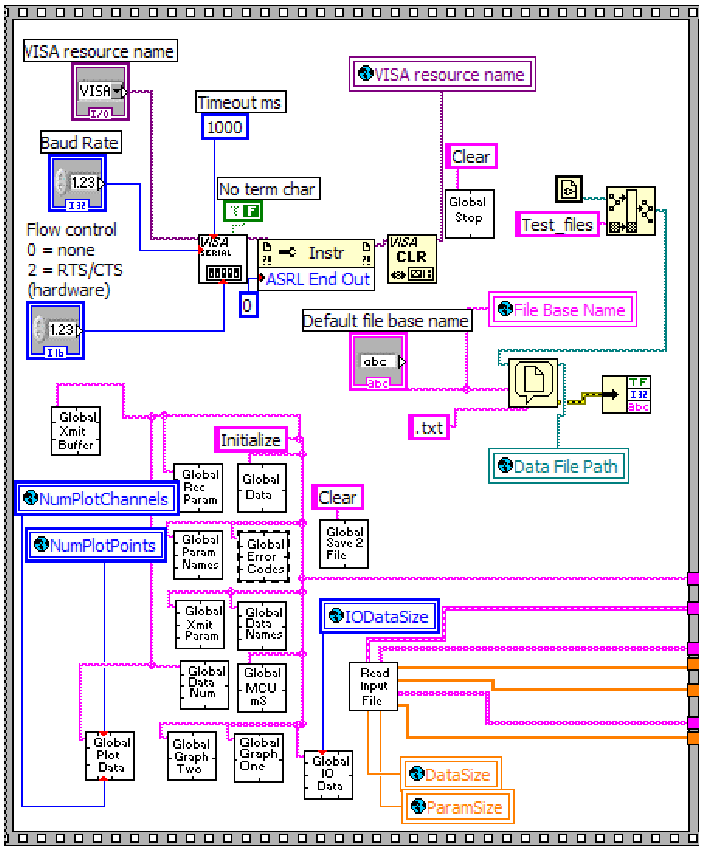
- The only inputs are from the Front Panel and are explained in the user's manual section of this report.

Outputs:

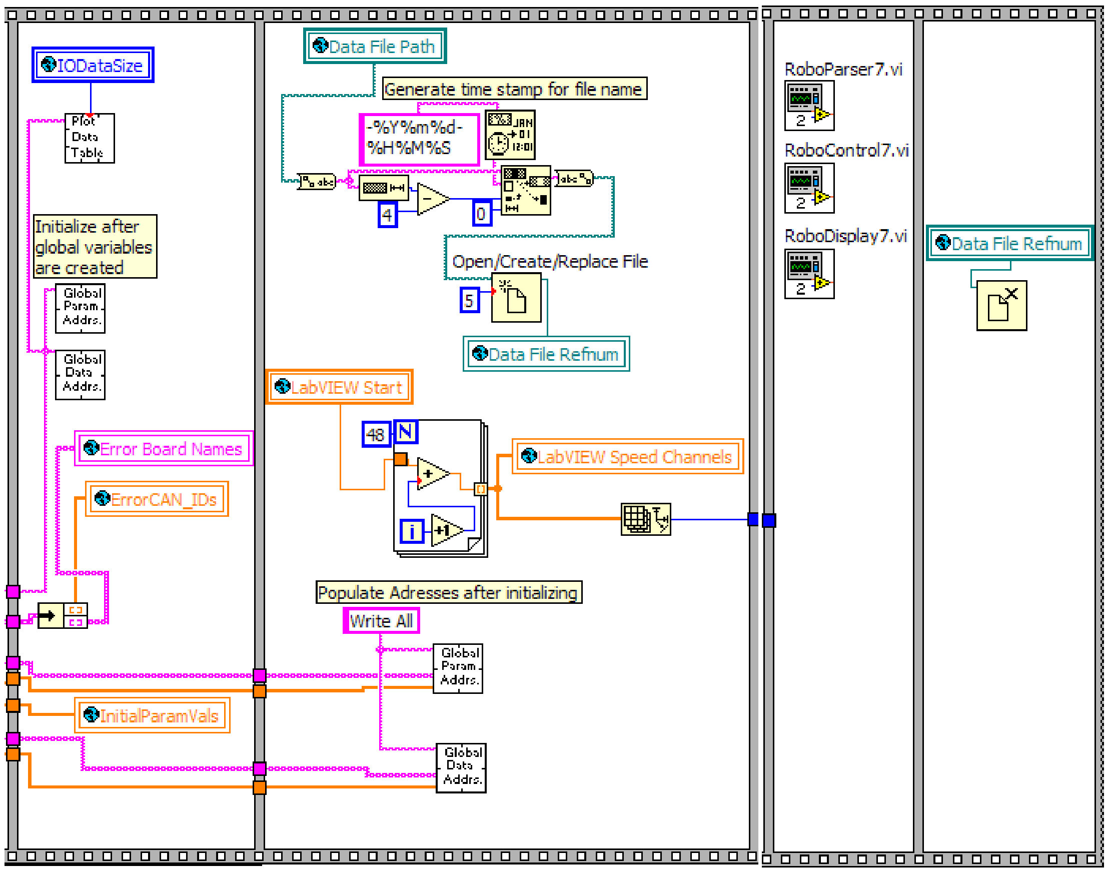
- N/A

Method:

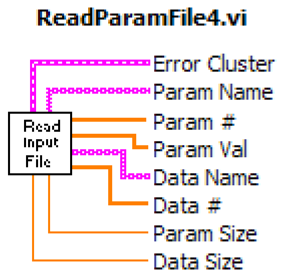
This program is segmented into a sequence structure consisting of five discrete sections. The first segment serves a number of functions. The first of which is to read in the hardware information to correctly configure the data rate and Bluetooth module settings. Additionally this section creates a default base file name that will preceed the data timestamp for all saved files. This section also initializes a number of global variables that will be used throughout various programs. These global variables include GlobalErrorCodes7.vi which calls a function to interpret the error_id.h and board_id.h files upon initialization. Furthermore the function ReadParamFile4.vi is called here which parses the can_id.h file and distributes the outputs to their respective global variables. The first step in the sequence structure is shown below.



The second step in the sequence structure initializes a series of global variables that rely on tasks from the first step to be completed. Specifically the <u>ReadParamFile4.vi</u> function must be complete and *GlobalDataSize.vi* and *GlobalParamSize.vi* must be initialized before proceeding. The third step writes the timestamp for the initial save file generated and also creates the LabVIEW speed channel list. The 4th step is responsible for calling the three main programs for the LabVIEW Robot Data Acquisition System <u>RoboDisplay7.vi</u>, <u>RoboParser7.vi</u>, and <u>RoboControl7.vi</u>.



ReadParamFile4.vi



Purpose:

The purpose of this program is to read in the Data and Parameters from the can_id.h file and separate the information into variables that are easy to work with in LabVIEW. This program only gets called once at the beginning of the RoboDAQ7.vi program.

Location:

- RoboDAQ7.vi

Key Variables:

- GlobalLabVIEWStart.vi – This saves the CAN ID of the first of the 48 LabVIEW speed channels.

Inputs:

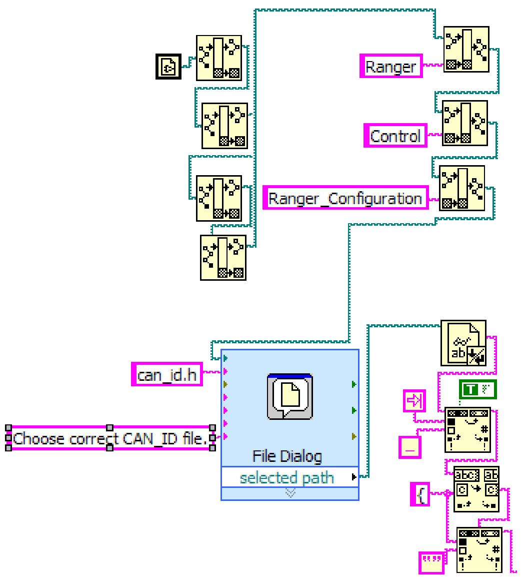
- The only input to this program is the can_id.h file that the user selects.

Outputs:

- Error Cluster (cluster of 2 arrays) – Contains one array of *ErrorCAN_IDs* and one array of the corresponding *Error Board Names*. Both of these get stored in *GlobalErrorCAN_ID.vi.*
- Param Name (array of strings) – Contains the names of each CAN ID channel that represents a parameter channel. Will be stored in *GlobalParamAddress.vi.*
- Param # (array of numeric) – Contains the CAN ID numbers of each data channel that represent parameters. Will be stored in *GlobalParamAddress.vi.*
- Param Val (array of numeric) – Contains the default values for all parameters which are hard coded into the can_id.h file. Will be stored in *GlobalInitialParamVals.vi.*
- Data Name (array of strings) – Contains the names of each CAN ID channel that represents a robot data channel. Will be stored in *GlobalDataAddress.vi.*
- Data # (array of numeric) – Contains the CAN ID numbers of each data channel that represents robot data. Will be stored in *GlobalDataAddress.vi.*
- Param Size (numeric) – Number of parameter channels used. Will be stored in *GlobalParamSize.vi*.
- Data Size (numeric) – Number of data channels used. Will be stored in *GlobalDataSize.vi.*

Method:

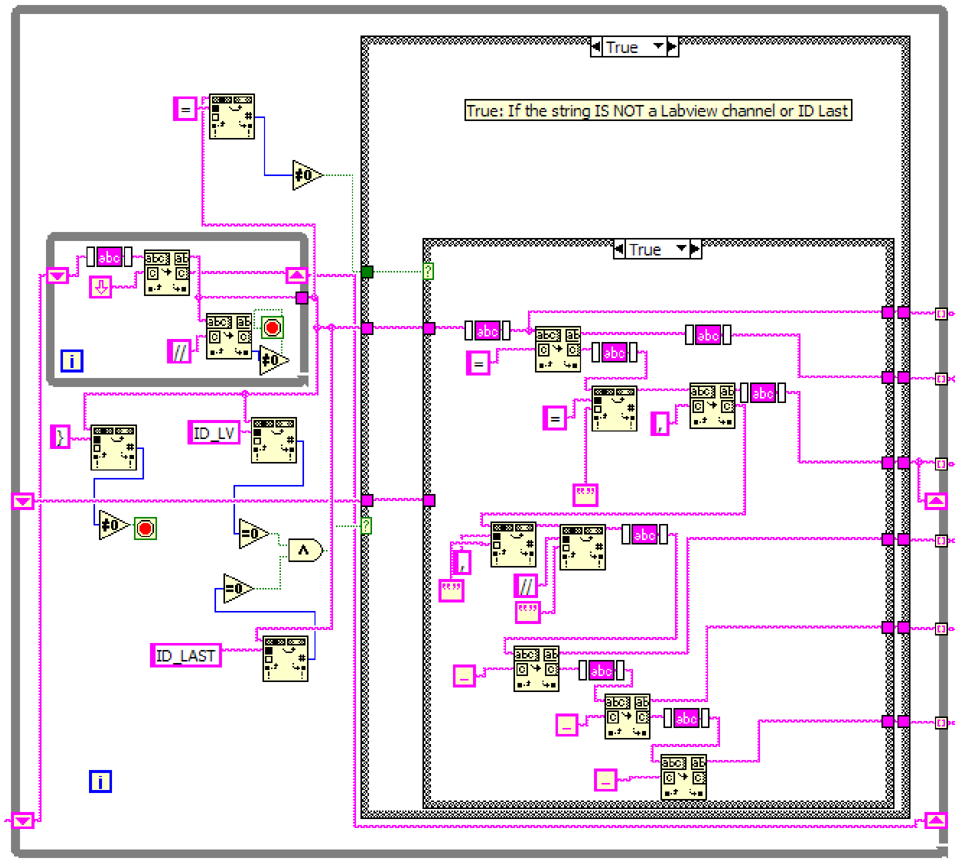
The program starts by finding the current folder location of the LabVIEW programs. It then uses this information to navigate to the correct folder which contains the default can_id.h file. The user may then choose the correct CAN ID file with a file dialog. The program then searches the selected file for the first "{" character and takes this to be the beginning of the CAN ID enumeration. The code which performs these steps is shown in the figure below



The program then begins to read the can_id.h file line by line searching for "Line Feed Constant" at the end of the string to denote a new line. All lines which begin with comments are entirely ignored. Lines that are not commented out are read into the LabVIEW program and the information in each line is parsed into its components. Each line should be of the format:

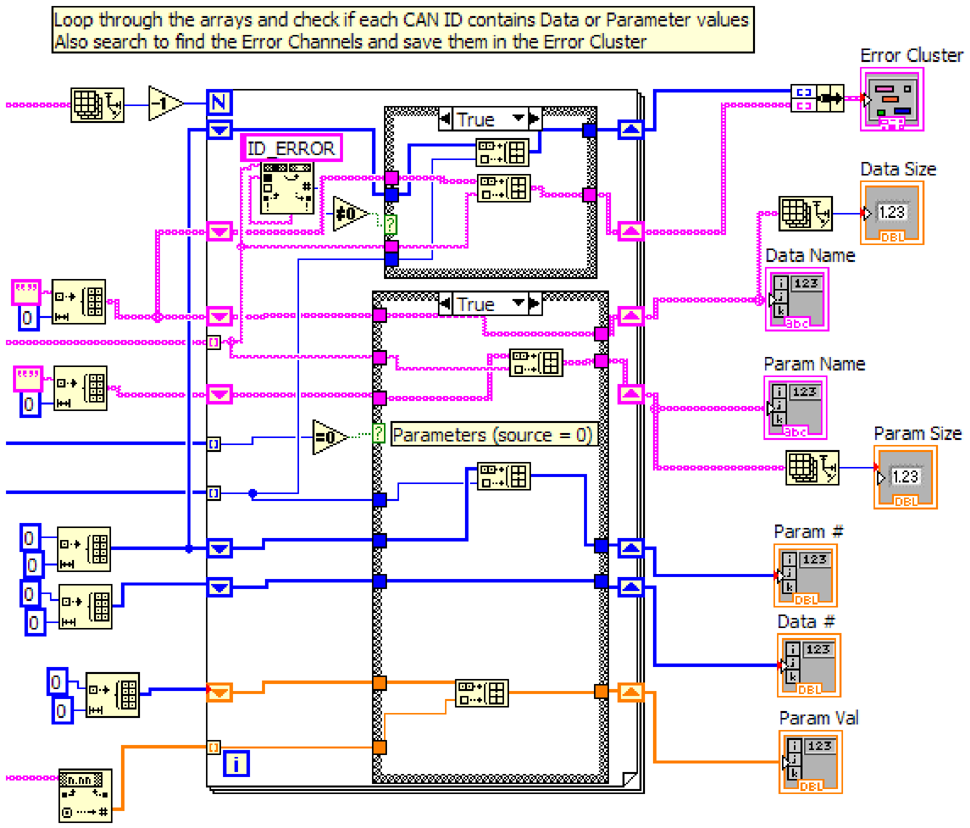
```
ID_BOARDNAME_DATANAME = <CANID>, // <InitialValue> <Source#> <Dest#>
```

Where the first string (with no spaces) is the name of the CAN ID channel, `<CANID` > would be replaced with an integer value of the CAN ID, "`<InitialValue>`" would be replaced by the initial default parameter value, and "`<Source#>`" is an integer representing the board from which the data is originating from. A `<Source#>` value of 0 represents that the CAN ID channel is a parameter channel. `<Dest#>` is not used by LabVIEW at the moment but it represents where information is being sent to. The program searches for spaces, "=", "//", and "," characters in order to determine where each piece of information is located. The code which separates the components of each line is shown below.

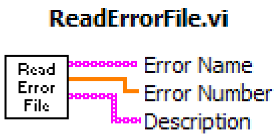


This program is also responsible for locating and numbering the LabVIEW speed channels which act to change the speed at which data is sent from the robot. It does this within the main loop shown above by searching for the string “ID_LV” within each line. Since LabVIEW channels are not initially given numbers in the can_id.h file, they are enumerated manually in LabVIEW.

Once the Program is finished parsing each line into individual strings of information, the strings are then converted into the correct format and sorted. The sorting is achieved by looping through the previously generated arrays and if the source number of that array element is equal to 0 then it is saved as a Parameter Channel, otherwise it is considered a Data Channel. Furthermore if the name of a given element contains ID_Error that information is saved as well and stored in an Error Cluster which will be used by the error handler. The code that sorts the CAN IDs into their appropriate catagories is shown below.


Loop through the arrays and check if each CAN ID contains Data or Parameter values
Also search to find the Error Channels and save them in the Error Cluster
ID_ERROR
True
True
Parameters (source = 0)
Error Cluster
Data Size
Data Name
Param Name
Param Size
Param #
Data #
Param Val

ReadErrorFile.vi



Purpose:

The purpose of this program is to read in the error names, descriptions, and numbers from the error_id.h file and separate the information into variables that are easy to work with in LabVIEW. This program only gets called once upon initialization of the *GlobalErrorCodes7.vi* global variable which occurs in RoboDAQ7.vi.

Location:

- *GlobalErrorCodes7.vi*

Key Variables:

- N/A

Inputs:

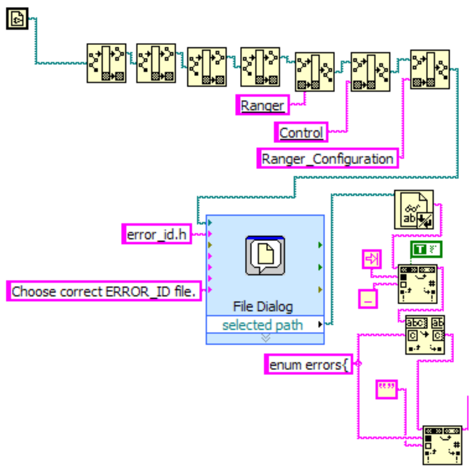
- The only input to this program is the error_id.h file that the user selects.

Outputs:

- Error Name (array of strings) – Contains an array of error names in the order in which they appear in the error_id.h file. This array is stored in the *GlobalErrorCodes7*.vi global variable and is accessed by using the “Read One” command and the “Error Name” node.
- Error Number (array of numeric) – Contains an array of error ID numbers in the order in which they appear in the error_id.h file. This array is stored in the *GlobalErrorCodes7*.vi global variable and is accessed implicitly by the index of the Error Name array.
- Description (array of strings) – Contains the description of each error in an array. This array is stored in the *GlobalErrorCodes7*.vi global variable and is accessed by using the “Read One” command and the “Description” node.

Method:

This program starts by locating the correct location of the error_id.h file based on the current location of the LabVIEW program and the relative SVN folder hierarchy. The user is then allowed to select the correct error_id.h file using a file dialog. The program then searches for the string “enum errors{“ in order to indicate the beginning of the error list. The code which does this is shown below.

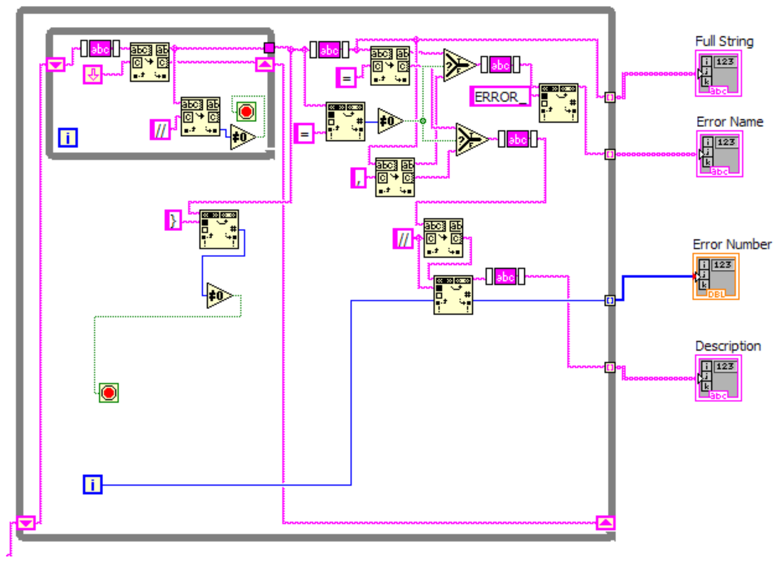


The code then loops through the error_id.h file reading it line by line while ignoring comment lines. When the code finds a non-commented line it selectively removes and stores the error name and error description based on the expected format shown here:

```
ERROR_NAME = 0, // <#Priority> <MOD>:  <Error description>
```

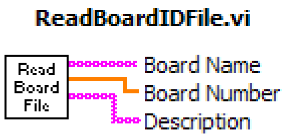
```
          Or

      ERROR_NAME, // <#Priority> <MOD>:  <Error description>
```

Everything except for the text after the comment and the error name is ignored. The error numbers are currently assigned manually by LabVIEW. It is assumed that this will remain an enumeration starting with 0 being the first ID value. The code that does this is shown below.



The code is terminated when it finds the "}" character, indicating the end of the enumeration, therefore this should not be located in comments for any reason.

ReadBoardIDFile.vi



Purpose:

The purpose of this program is to read in the board names, and corresponding numbers from the board_id.h file and separate the information into variables that are easy to work with in LabVIEW. This program only gets called once upon initialization of the *GlobalErrorCodes7.vi* global variable which occurs in RoboDAQ7.vi.

Location:

- *GlobalErrorCodes7.vi*

Key Variables:

- N/A

Inputs:

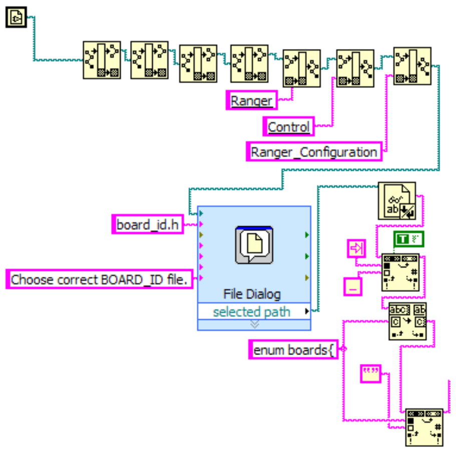
- The only input to this program is the board_id.h file that the user selects.

Outputs:

- Board Name (array of strings) – Contains an array of board names in the order in which they appear in the board_id.h file. This information gets stored in the *GlobalBoardIDs.vi* global variable under "Board Names".
- Board Number (array of numeric) – Contains an array of board numbers in the order in which they appear in the board_id.h file. This information gets stored in the *GlobalBoardIDs.vi* global variable under "Board ID #s".
- Description (array of strings) – This output is unused. If there were any comments in the board_id.h file they would be sent to this output.

Method:

This program starts by locating the correct location of the board_id.h file based on the current location of the LabVIEW program and the relative SVN folder hierarchy. The user is then allowed to select the correct board_id.h file using a file dialog. The program then searches for the string "enum boards{" in order to indicate the beginning of the board list. The code which does this is shown below.



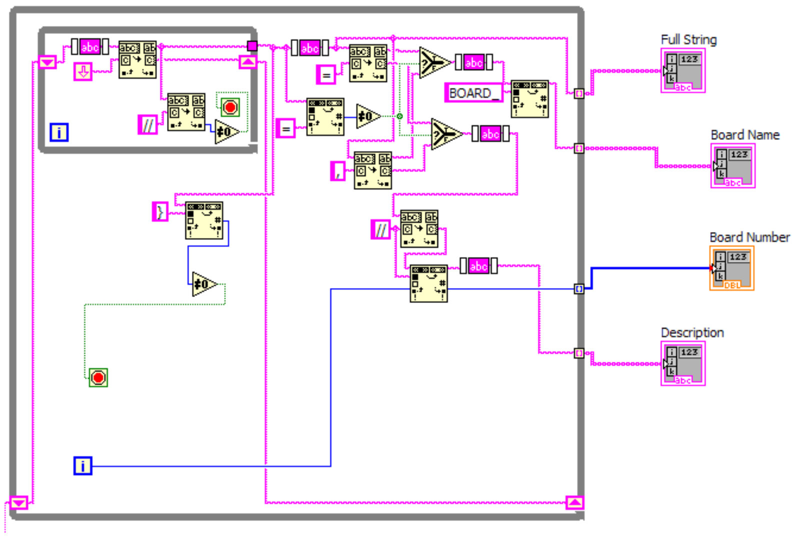
The code then loops through the board_id.h file reading it line by line while ignoring comment lines. When the code finds a non-commented line it selectively removes and stores the board name based on the expected format shown here:

```
BOARD_NAME = 0,
```

Or

```
        BOARD_NAME,
```

Everything except for the board name is ignored. The board numbers are currently assigned manually by LabVIEW assuming that the assignment of board numbers will remain an enumeration starting with 0. The code that does this is shown below.



The code is terminated when it finds the "}" character, indicating the end of the enumeration, therefore this should not be located in comments for any reason.

RoboControl7.vi

This program runs in parallel with RoboDisplay7.vi and was written by Jason Cortell and is largely for sending data to the robot from the LabVIEW system. This report does not cover this program.

RoboParser7.vi

This program runs in parallel with RoboDisplay7.vi and was written by Jason Cortell and is responsible for reading in the data from the robot and sending it to the rest of the LabVIEW program. This report does not cover this program.

RoboDisplay7.vi

This program is responsible for running the user interface and is comprised of many sub-programs and sections. In order to better explain how the code works I have broken it down into its constituent components. Note that not everything will be explained step by step however a general understanding of each part should be attained through this section.

Initializations

Purpose:

The purpose of this section is to initialize any local variables upon startup of the *RoboDisplay7.vi* program. This will assure that all fields are set to their default values instead of having saved older information in them. This is only called once before entering the main while loop of the program.

Location:

- RoboDisplay7.vi – everything to the left of the main while loop

Key Variables:

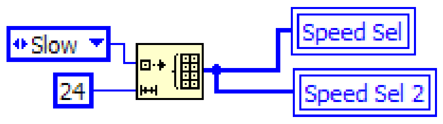
- There are a number of initialized variables, it is best to go to the LabVIEW code and either right click and select "find" or double click to find out what the variable is referring to.

Key LabVIEW Functions:

- Initialize Array – This is used to initialize most of the local and global variables.

Method:

Fairly straightforward, assign the data type, value, and size to each of the local variable arrays using the *Initialize Array* function. A sample assignment of the initial speed select enumeration values is shown below.



Welcome Tab

Purpose:

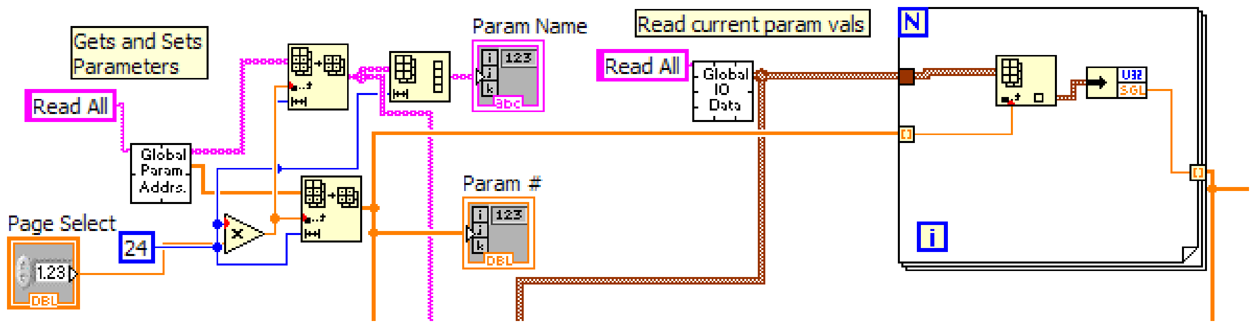
This tab is a place holder for the Welcome Screen on the front panel. It contains no block diagram code.

Parameters Tab

Purpose:

The purpose of this tab is to display robot parameter values and provide a means for the user to modify them in an easy way. I have broken this section down into a number of categories in order to make it easier to understand.

Read in Parameters from robot



Purpose:

The purpose of this section is to read the parameter names and parameter values from the robot and display them for the user.

Location:

- RoboDisplay7.vi –Parameters Tab

Key Variables:

- *GlobalParamAddress.vi* – Contains all of the parameter names in order of increasing CAN ID number
- *GlobalIOData7.vi* – Contains all parameter and data values being sent from the robot.

Key LabVIEW Functions:

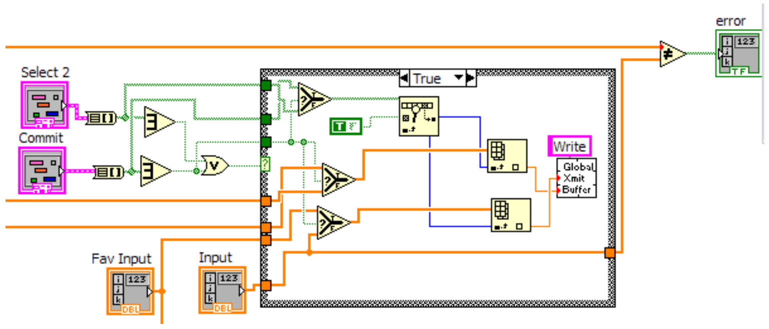
- Unbundle – needed to separate the cluster components from *GlobalIOData.vi*

Method:

Use the “Read All” command on the *GlobalParamAddress7.vi* global variables to get all the parameter names. Send the appropriate selection of these names (based on page number and number of slots displayed) to the Param Name indicator array.

Use the “Read All” command on the *GlobalIOData7.vi* to get the current values of the parameters from the robot. Loop through this array of clusters and unbundle each one and index them using a for loop. Send the value to the Param Value indicator.

Assign Parameter Value to Robot



Purpose:

The purpose of this section is to send a desired parameter value to a buffer that will in turn send the new value to the robot.

Location:

- RoboDisplay7.vi –Parameters Tab

Key Variables:

- *GlobalTransmitBuffer7.vi* – Used to send values to a specified CAN ID on the robot.

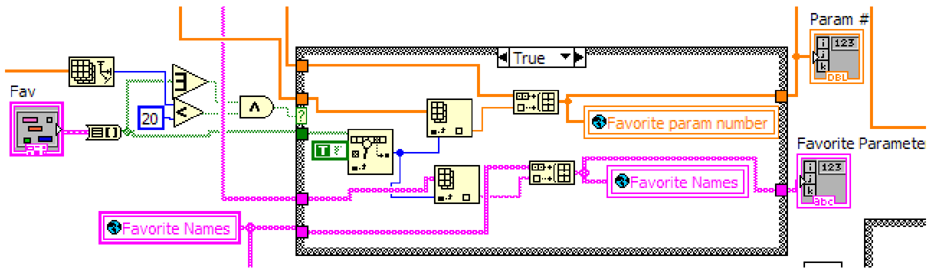
Key LabVIEW Functions:

- Cluster to Array – converts a cluster (in this case a cluster of Booleans) to an array
- Or Array Elements – Returns true if any Boolean in an array is true
- Select – Returns one value if true and a different value if false.

Method:

Check to see if either a new parameter or favorite parameter value has been committed by the user. If a value was submitted find the corresponding parameter CAN ID by searching the Boolean array. Send the committed value as well as the CAN ID number by using the "Write" command in the *GlobalTransmitBuffer*.vi global variable.

Add Favorite



Purpose:

The purpose of this section is to add a parameter to the favorites list. This list can set up by the user to contain the parameters that they use the most.

Location:

- RoboDisplay7.vi –Parameters Tab

Key Variables:

- *Favorite Names* – an array of strings containing the names of the favorite parameters
- *Favorite Param Number* – an array of numeric containing the CAN IDs of the favorite parameters

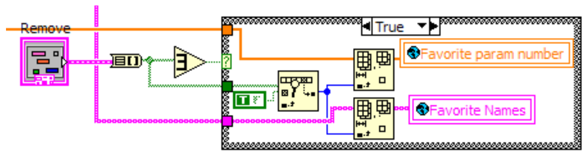
Key LabVIEW Functions:

- Build Array – takes an existing array and concatenates a new element to the end.

Method:

Check to see if a parameter has been selected as a favorite by checking the **Fav** cluster. If so add the name of this parameter to the *Favorite Names* global variable and the CAN ID number to the *Favorite Param Number* global variable.

Remove Favorite



Purpose:

The purpose of this section is to remove a parameter from the favorites list.

Location:

- RoboDisplay7.vi –Parameters Tab

Key Variables:

- *Favorite Names* – an array of strings containing the names of the favorite parameters
- *Favorite Param Number* – an array of numeric containing the CAN IDs of the favorite parameters

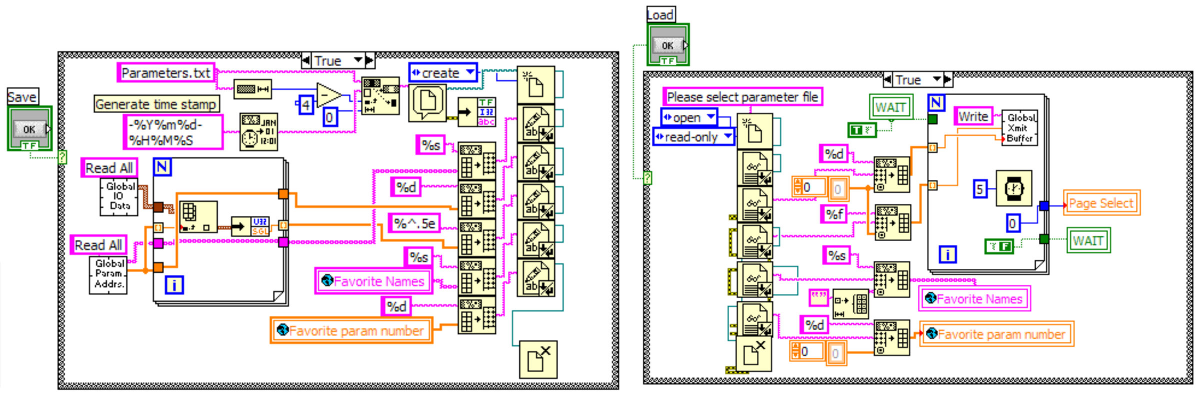
Key LabVIEW Functions:

- Delete from Array – takes an existing array and removes the element in the specified index location.

Method:

Check to see if a favorite parameter has been selected to be removed by checking the **Remove** cluster. If so, delete the appropriate elements in the Favorites global variables by using the *Delete From Array* function in LabVIEW.

Save/Load Parameters



Purpose:

The purpose of this section is to remove a parameter from the favorites list.

Location:

- <u>RoboDisplay7.vi</u> –Parameters Tab

Key Variables:

- *Favorite Names* – an array of strings containing the names of the favorite parameters
- *Favorite Param Number* – an array of numeric containing the CAN IDs of the favorite parameters
- *GlobaTransmitBuffer*.vi – a buffer for sending information to the robot.

Key LabVIEW Functions:

- Array to Spreadsheet String – turns an array of a specified type into an array of text strings that can be easily written to a text file
- Write to Text File – writes a line of text and then begins a new line.
- Read From Text File – reads a line of text from a specified file at the current pointer location.
- Format Date/Time String – takes a timestamp input and returns the information in a more useful format as specified by the user.
- Spreadsheet String to Array – takes a string and separates it into components based on a delimiting option.

Method:

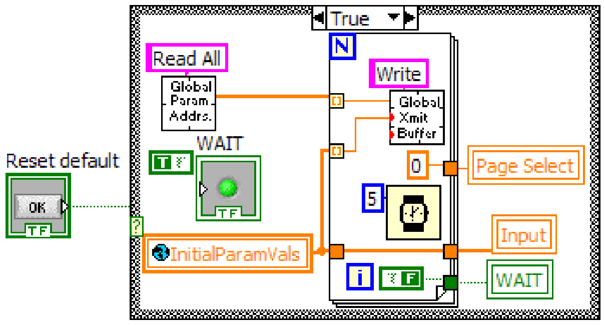
To save a parameter file, generate a new time stamp at the time the **Save** button is pressed, this will be inserted into the default file name which is "Parameters.txt." A new blank file must be opened. The file is then written line by line in a tab delimited format as follows:

Row 1 – an array of strings retrieved from the parameter names saved in *GlobalParamAddress.vi*
Row 2 – an array of doubles retrieved from the parameter CAN IDs saved in *GlobalParamAddress.vi*
Row 3 – an array of floating point parameter values read from GlobalIOData.vi for all of the CAN IDs listed in *GlobalParamAddress.vi.*
Row 4 – an array of strings retrieved from the parameter names saved in *Favorite Names*
Row 5 – an array of doubles retrieved from the parameter CAN IDs saved in *Favorite Param Number*

The file is then closed after all 5 rows of data have been written.

To load a parameter file the user selects an appropriate file and it is then read line by line according to the above format. The information is interpreted and saved into the locations that they came from with one exception; the parameter values have to be sent to the transmit buffer. This is done at a controlled rate in order to not overwhelm the transmit buffer. During this time a red light displays on the user interface telling them to wait for a moment.

Reset to default Parameters



Purpose:

The purpose of this section is to reset the parameter values on the robot to the default parameters specified by the can_id.h file that was read upon initialization.

Location:

- RoboDisplay7.vi –Parameters Tab

Key Variables:

- *InitialParamVals* – an array of the default values of each parameter.
- *GlobalParamAddress.vi* – contains the parameter names and CAN IDs

- *GlobaTransmitBuffer*.vi – a buffer for sending information to the robot.

Key LabVIEW Functions:

- Wait (ms) – waits until a specified amount of milliseconds before continuing.

Method:

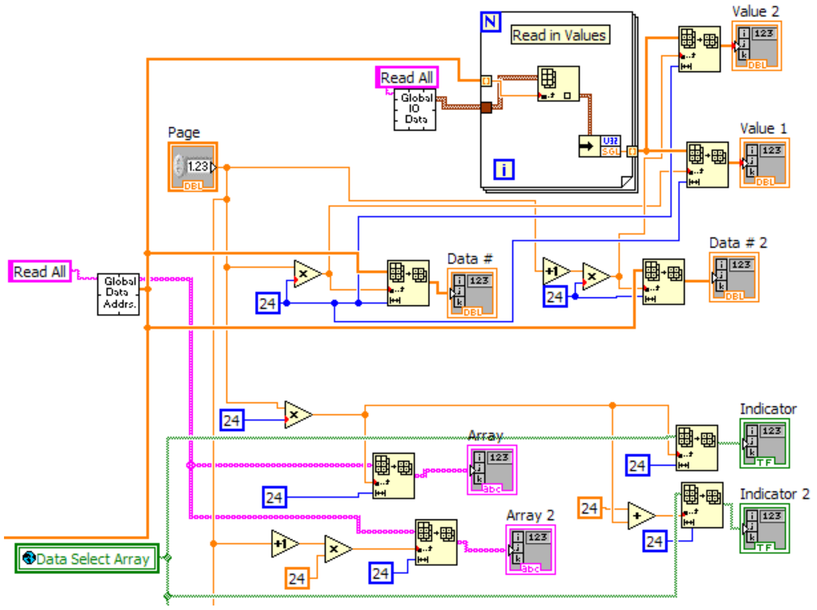
Loop through all CAN ID values corresponding to parameters by reading the *GlobalParamAddress* global variable and read the corresponding default parameter value from *InitialParamVals*. Send these values and CAN IDs one pair at a time every 5 milliseconds to the *GlobalTransmitBuffer*.

Data Select Tab

Purpose:

The purpose of this tab is to allow the user to select and deselect data channels of interest. The user may also view current values of each data channel here as a digital numeric output. I have once again broken the code into sections for easier understanding.

Read and Display Data Values



Purpose:

The purpose of this section is to read in the current data values and display them for the user. Additionally the user may select up to 48 data channels that they are potentially interested in viewing on a graph or saving at a faster data rate for later analysis.

Location:

- RoboDisplay7.vi –Data Select Tab

Key Variables:

- *GlobalDataAddress7.vi* – contains the data channel names and CAN IDs
- *GlobalIOData7.vi* – Contains all parameter and data values being sent from the robot.

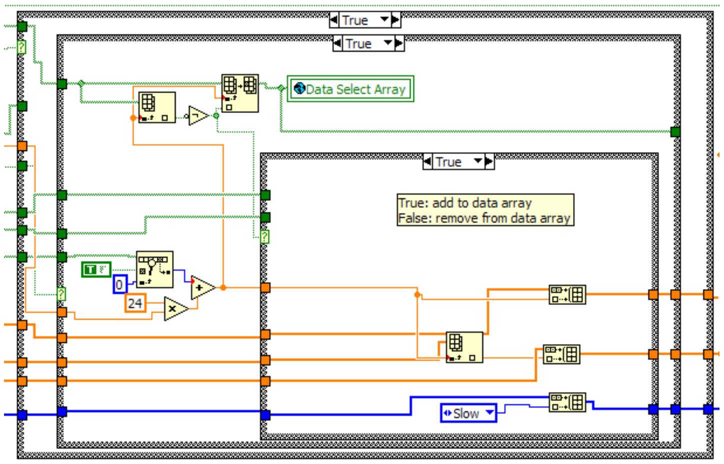
Key LabVIEW Functions:

- Nothing particularly complicated

Method:

Read all data value information from *GlobalIOData.vi* and the names and CAN IDs from *GlobalDataAddress.vi* and select the appropriate subset of this data based on the page number and number of available slots (in this case 2 sets of 24 slots) on the front panel. Send the information to the appropriate indicators. Additionally in this section the data channels that have been selected are indicated with a Boolean array of green lights on the front panel. The Boolean values are assigned in the "Add/Remove Data Channels" section but they are displayed here through use of the *Data Select Array* global variable.

Add/Remove Data Channels



Purpose:

The purpose of this section is to select or deselect data channels that will be sent to the Data Display Tab.

Location:

- RoboDisplay7.vi –Data Select Tab

Key Variables:

- *GlobalDisplayAddress* – A global variable containing the following information

- *Display canID* – contains an array of CAN IDs that will be displayed in the Data Display section.
- *Display Address* – an array that contains the index of the desired CAN IDs in the *GlobalDataAddress.vi* global variable.

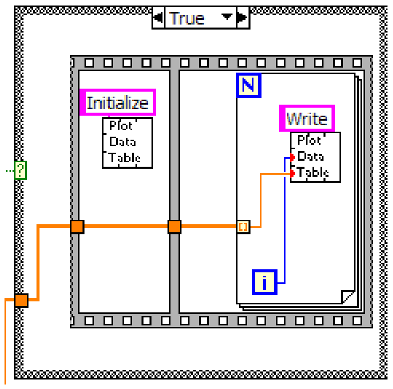
Key LabVIEW Functions:

- Split 1D Array – divides an array at the given index and returns both portions
- Delete From Array – deletes an array element at the given index
- Search 1D Array – searches an array for a given element and returns the index

Method:

This section uses 3 nested case structures to account for all possible scenarios that could occur. The outermost case structure is true if there are less than 48 data channels selected and false otherwise. If there are 48 or more data channels selected the case structure will ensure that no more data channels can be added, only a deselect option will be available. The next case structure then checks whether or not a select/deselect button has been pressed by the user. If this false nothing happens if this is true it then finds the CAN ID of the data channel selected and proceeds to the final case structure. This final case structure then decides whether the CAN ID has already been selected, if this is the case then it removes the CAN ID otherwise it will add the CAN ID to the list of selected data channels. The outputs of this are sent to the Update Plot Data Table section.

Update Plot Data Table



Purpose:

The purpose of this section is to update the Plot Data Table global variable. This variable enables plotting of the CAN IDs sent to it.

Location:

- RoboDisplay7.vi –Data Select Tab

Key Variables:

- *GlobalPlotDataIDTable7.vi* - contains the CAN IDs of the data channels that need to have a time history saved for the purposes of graphing.

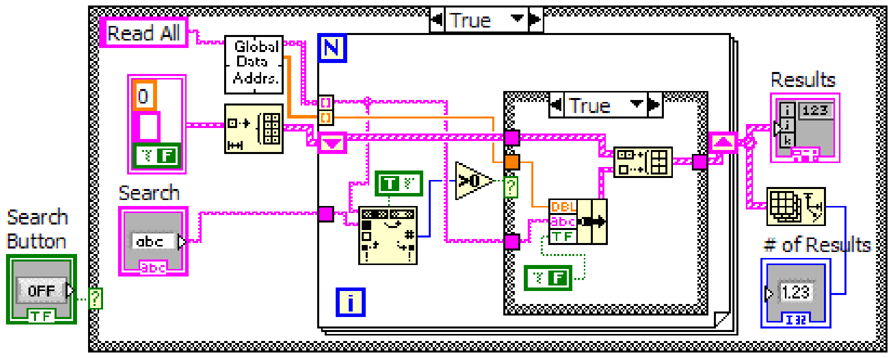
Key LabVIEW Functions:

- Nothing particularly complicated

Method:

When a new data channel is selected or deselected the *Plot Data Table* global variable is first reset by using the "Initialize" function call. Then each data channel is sent to the global variable one by one using the "Write" command.

Search Data Channels



Purpose:

The purpose of this section is to search all data channel names for a string that is input by the user.

Location:

- RoboDisplay7.vi –Data Select Tab

Key Variables:

- *GlobalDataAddress7.vi* – contains the data channel names and CAN IDs

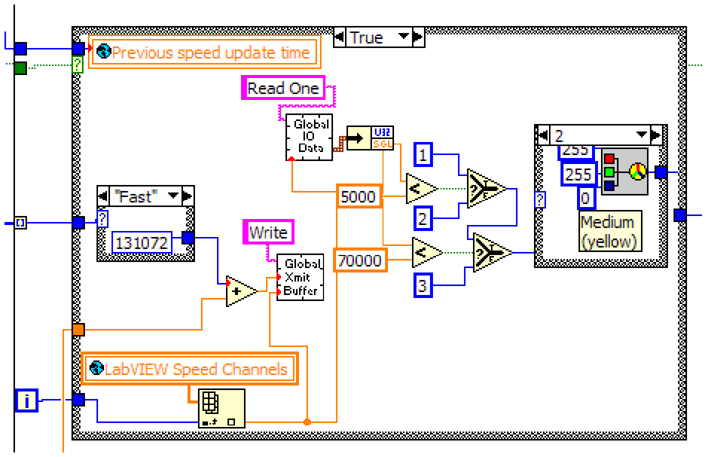
Key LabVIEW Functions:

- Search and Replace String – searches for a given input string and replaces it with a desired string. Also returns whether or not an instance of the string was found (which is the useful part for this section)
- Bundle – groups different types of data together into a cluster.

Method:

When a user enters a string and presses the search button the code reads all of the data channel names and loops through them one by one. Each time a match is found within a name that data channel CAN ID and name are added to a bundle. When the loop is finished the bundle is output to the Results display cluster for the user to review.

Data Display Tab



Purpose:

The purpose of this tab is to handle the selection and assignment of data speeds as well as displaying the channels that may be plotted. The graphing itself is done outside of the tab structure so that it can be run continuously in the background.

Location:

- RoboDisplay7.vi –Data Display Tab

Key Variables:

- *GlobalIOData7.vi* – contains all parameter and data values being sent from the robot.
- *GlobaTransmitBuffer*.vi – a buffer for sending information to the robot.
- *LabVIEW Speed Channels* – contains a list of the CAN IDs corresponding to the speed channels. These channels are used to set the data speeds of the selected data channels.
- *Display Address* – an array that contains the index of the desired CAN IDs in the *GlobalDataAddress.vi* global variable.

Key LabVIEW Functions:

- RGB to color.vi – converts red, green, and blue values to the combined output color.
- Case Structure – The case structure used here uses enumerated values for Slow, Medium, and Fast instead of just True and False.

Method:

The radio buttons on the user interface output a value of 0, 1, or 2 corresponding to Slow, Medium, or Fast respectively. Based on these user inputs a speed values is selected and in order to set the speed you must assign a value of

AssignedValue = SPEEDVAL + CAN_ID#

to the appropriate LabVIEW speed channel corresponding to the position of the data in the data table where SPEEDVAL is:

0 for slow
65536 for medium
131072 for fast

After the assigned value is determined it is sent to the *GlobalTransmitBuffer* to be updated on the robot. The program then reads back the value that is on the robot and interprets the value to a speed based on its range. Then the results are displayed using a Color Box indicator where:

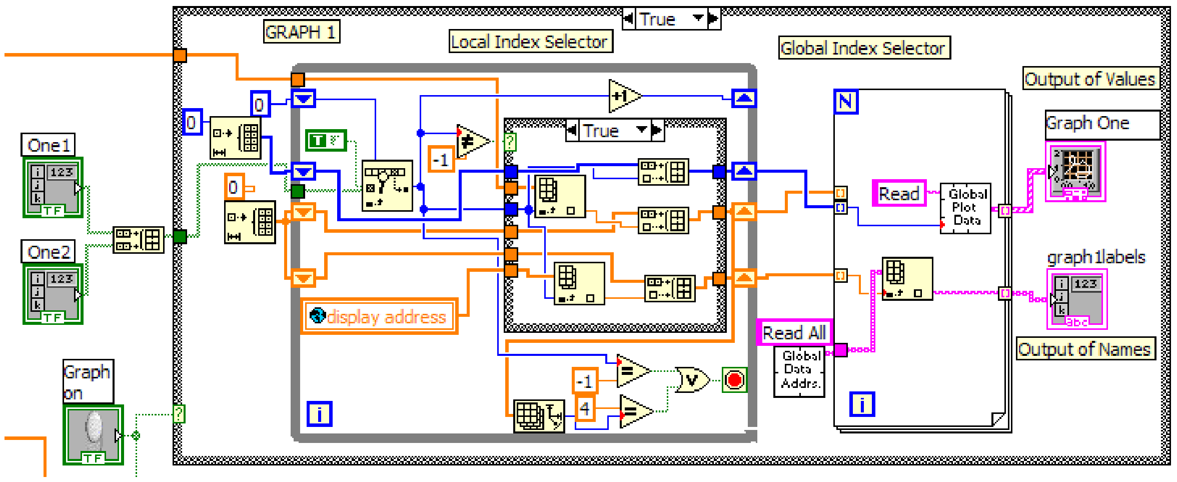
Red = Slow
Yellow = Medium
Slow = Fast

Errors Tab

Purpose:

As of right now this tab does nothing in the block diagram, it is only there to hold the place for the front panel error display. All error handling is done outside of the tab structure so that errors are continuously monitored. See the Error Handling section for more details on everything related to errors.

Graphs



Purpose:

The purpose of this section is to plot up to 4 variables simultaneously on one graph. There is a separate case structure for Graph 1 and for Graph 2 but each of them performs identical tasks.

Location:

- RoboDisplay7.vi - outside of tabbed region in main while loop

Key Variables:

- *GlobalPlotData7.vi* – contains the time history of the data channels listed in *GlobalPlotDataIDTable7.vi* .
- *GlobalPlotDataIDTable7.vi* - contains the CAN IDs of the data channels that need to have a time history saved for the purposes of graphing.
- *GlobalDataAddress7.vi* – contains the data channel names and CAN IDs
- *Display Address* – an array that contains the index of the desired CAN IDs in the *GlobalDataAddress7.vi* global variable.

Method:

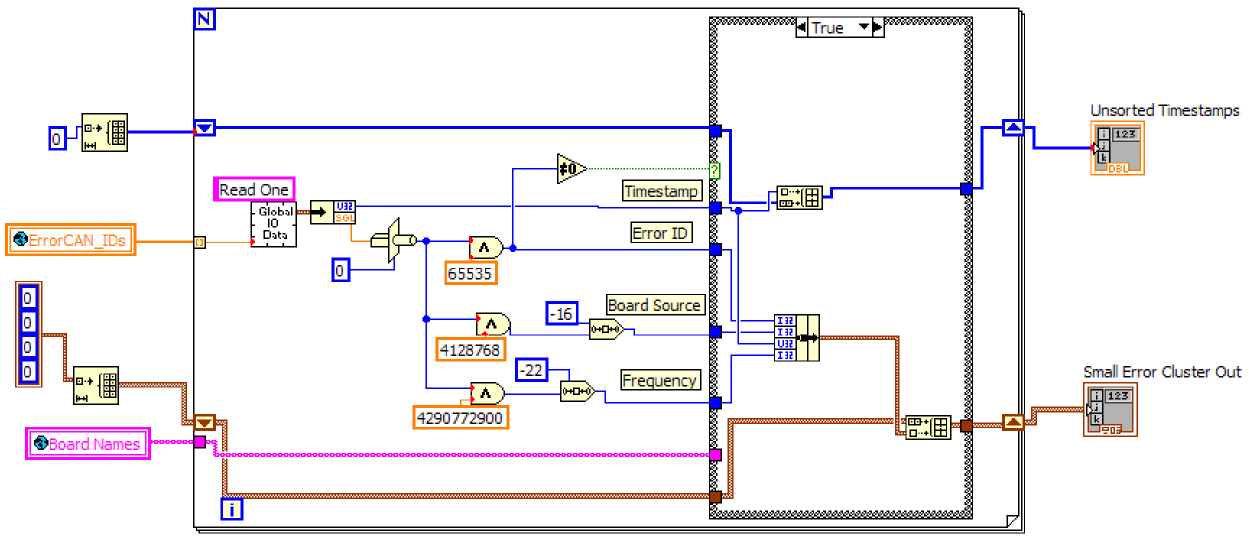
This section begins by reading in the CAN IDs of the data channels which are in the Data Select Tab. Next the program checks to see which of these data channels have been selected and extracts the *Display Address* value of that selection. This *Display Address* can then be used to retrieve the Data Name from *GlobalDataAddress.vi*. The index of the selected element in the*GlobalDataNum7.vi* global variable is used to access *GlobalPlotData.vi* . Note: *GlobalDataNum7.vi* contains the same values as *GlobalPlotDataIDTable7.vi* except it is not used by other programs created by Jason Cortell. A maximum of 4 data channels may be plotted at any given time.

Error Handling

Purpose:

The purpose of this section is to read, organize, and display error messages for the user. The error handling system consists of multiple sub VIs which I will discuss individually in the order of their use.

ErrorHandlerREADALL.vi



Purpose:

The purpose of this section is to read in the values on each of the error channels and interpret them into the Error ID, Board Source, and Frequency information that is stored within each value.

Location:

- ErrorHandlerREADALL.vi - Called from RoboDisplay7.vi outside of tabbed region in main while loop.

Key Variables:

- *GlobalIOData7.vi* – contains data and parameter values for each CAN ID.
- *ErrorCAN_IDs* – contains an array of CAN IDs that correspond to the channels on which error messages are sent.

Key LabVIEW Functions:

- Type Cast – Casts data to the type specified.
- Logical Shift – Shifts a value by the number of bits specified

Method:

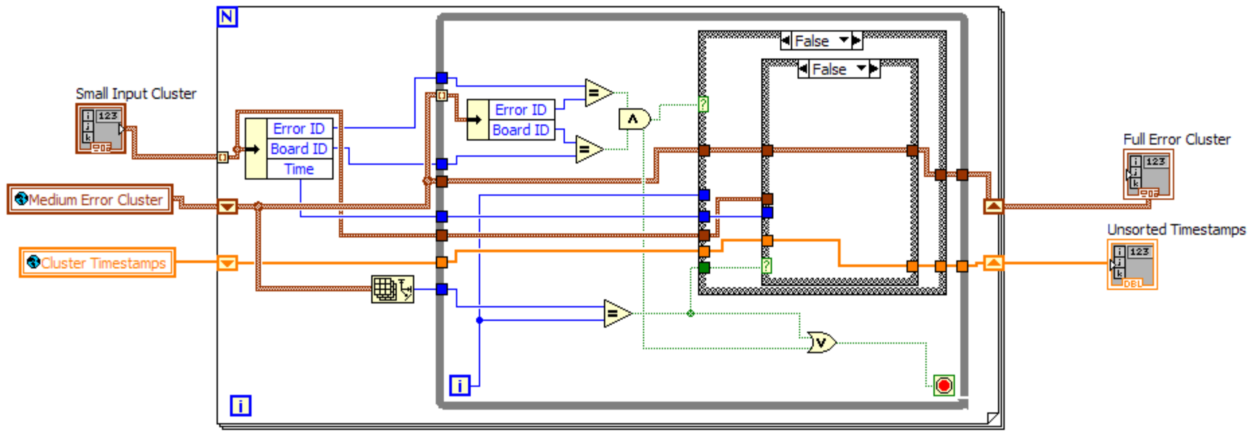
The program reads in the values on each of the error channels individually and interprets them according to the following scheme:

(bit 0 = Least Significant Bit)

bits 0 to 15 (16 bits total): Error code
bits 16 to 21 (6 bits total): Board code
bits 22 to 31 (10 bits total): Error frequency

These values are then sent on to the ErrorHandlerUpdate.vi program through the “Small Error Cluster Out” indicator.

ErrorHandlerUpdate.vi



Purpose:

The purpose of this program is to take the most recently received errors and determine whether they are new and need to be added or existing and need to be updated.

Location:

- ErrorHandlerUpdate.vi - Called from RoboDisplay7.vi outside of tabbed region in main while loop.

Key Variables:

- *ClusterTimestamps* – contains the cluster of timestamps of the existing errors
- *MediumErrorCluster* – contains a cluster of error information that can be passed around in the error handler

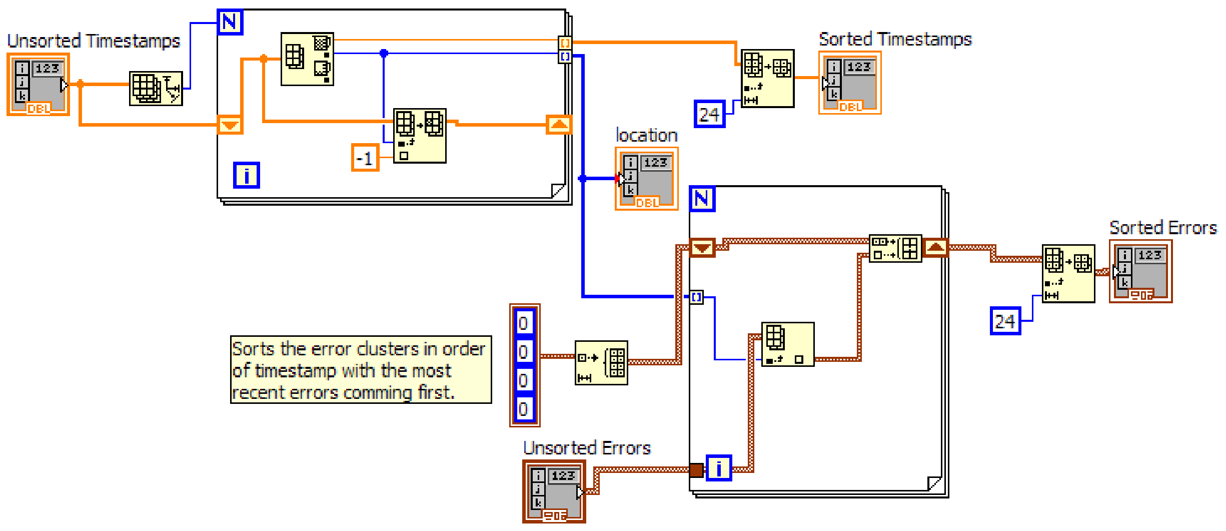
Key LabVIEW Functions:

- Unbundle by Name – takes a cluster and extracts only the named elements.

Method:

New error information is passed in from ErrorHandlerREADALL.vi . The program then checks if an error already exists by looping through and comparing new error and board numbers to the current list. If an error does exist but has occurred again it updates the current value of the timestamp. If an error does not exist then it is added to the error list along with its timestamp Error information is then sent out

ErrorClusterSort2.vi


Unsorted Timestamps
Sorted Timestamps
location
Sorted Errors
Unsorted Errors
-1
24
24
Sorts the error clusters in order of timestamp with the most recent errors comming first.

Purpose:

The purpose of this program is to sort the newly formed list of errors into chronological order starting with the most recent.

Location:

- ErrorClusterSort2.vi - Called from RoboDisplay7.vi outside of tabbed region in main while loop.

Key Variables:

- *None*

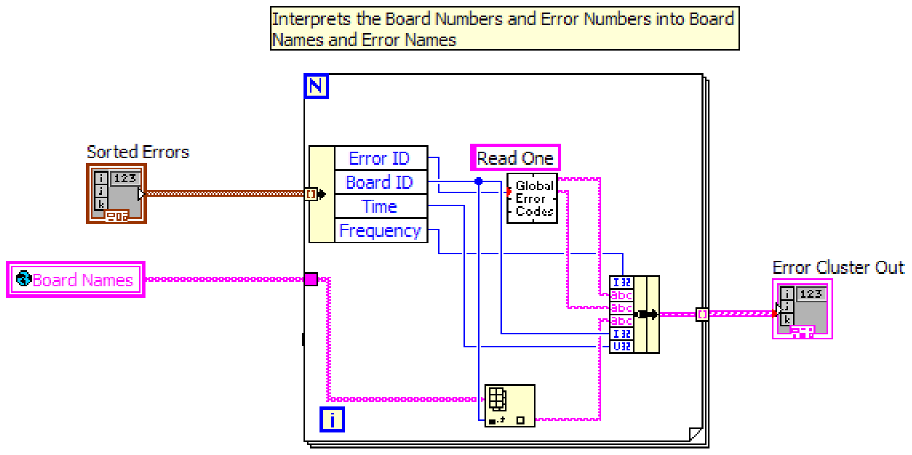
Key LabVIEW Functions:

- Array Min & Max – finds the minimum and maximum values and returns their value and index.

Method:

This program reads in the unsorted timestamps and corresponding errors from ErrorHandlerUpdate.vi. The program then loops through all of the timestamps and locates the maximum value. This value is then removed and replaced with a value of -1 while the index of the most recent timestamp (the highest) is saved. Once all of the indices are saved in order of highest to lowest they are used to map the unsorted errors to a sorted error array.

ErrorHandlerLabel.vi



Purpose:

The purpose of this program is to assign board names, error names, and error descriptions to the error codes received.

Location:

- ErrorHandlerLabel.vi – called from RoboDisplay7.vi outside of tabbed region in main while loop.

Key Variables:

- *Board Names* – contains a list of the board names sorted by board ID number.

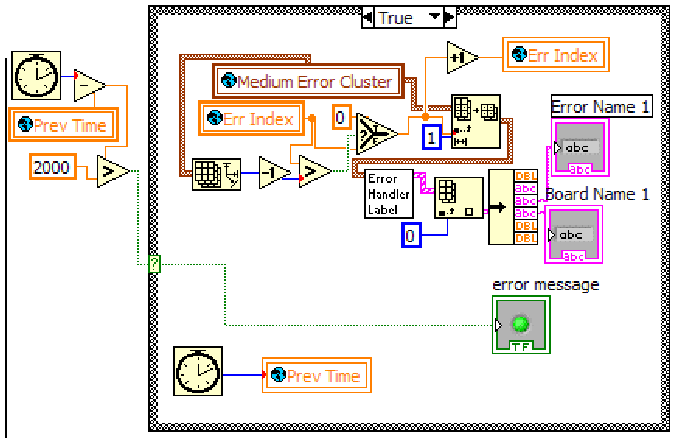
Key LabVIEW Functions:

- Unbundle by Name – ungroups a cluster of variables by name.
- Bundle – groups various variables together in a cluster.

Method:

This program reads in the sorted errors from ErrorClusterSort2.vi and assigns board names, error names, and error descriptions to the error based on the Error ID, and Board ID numbers. These are then passed out to the front panel for the user.

Quick Error Display



Purpose:

The purpose of this program is to periodically display errors on the front panel in the Quick Error Display bar.

Location:

- RoboDisplay7vi – outside of tabbed region in main while loop.

Key Variables:

- *Medium Error Cluster* – a cluster of error codes, frequencies, board ID numbers, and timestamps.

Key LabVIEW Functions:

- None

Method:

This program is called every 2 seconds to update a display on the front panel which cycles through each of the active errors. Upon being called the program starts at the 1st element in the *Medium Error Cluster* global variable and displays the error name and board name. Every consecutive time this function is called it increments the value of *Err Index*, which corresponds to the element of interest in the error cluster until it reaches the end. Upon reaching the end of the error cluster the *Err Index* value resets to 0.