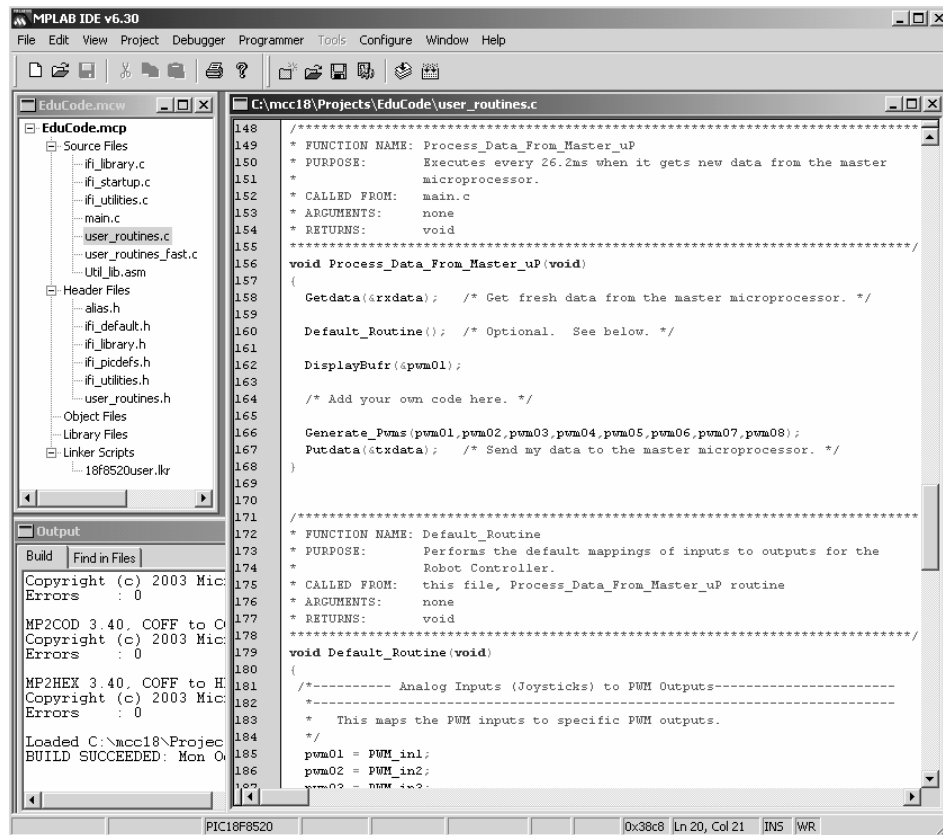


Innovation First, Inc.

2004 Programming Reference Guide



The screenshot displays the MPLAB IDE v6.30 interface. The main window shows the source code for `C:\mcc18\Projects\EduCode\user_routines.c`. The code includes comments and function definitions for `Process_Data_From_Master_uP` and `Default_Routine`. The `Process_Data_From_Master_uP` function is called from `main.c` and generates PWM signals. The `Default_Routine` function performs default mappings of inputs to outputs for the robot controller.

```
148 /*****
149 * FUNCTION NAME: Process_Data_From_Master_uP
150 * PURPOSE:      Executes every 26.2ms when it gets new data from the master
151 *               microprocessor.
152 * CALLED FROM:  main.c
153 * ARGUMENTS:    none
154 * RETURNS:      void
155 *****/
156 void Process_Data_From_Master_uP(void)
157 {
158     Getdata(&rxdata); /* Get fresh data from the master microprocessor. */
159
160     Default_Routine(); /* Optional. See below. */
161
162     DisplayBufx(&pwm01);
163
164     /* Add your own code here. */
165
166     Generate_Pwms(pwm01,pwm02,pwm03,pwm04,pwm05,pwm06,pwm07,pwm08);
167     Putdata(&txdata); /* Send my data to the master microprocessor. */
168 }
169
170
171 /*****
172 * FUNCTION NAME: Default_Routine
173 * PURPOSE:      Performs the default mappings of inputs to outputs for the
174 *               Robot Controller.
175 * CALLED FROM:  this file, Process_Data_From_Master_uP routine
176 * ARGUMENTS:    none
177 * RETURNS:      void
178 *****/
179 void Default_Routine(void)
180 {
181     /*----- Analog Inputs (Joysticks) to PWM Outputs-----
182     *-----
183     * This maps the PWM inputs to specific PWM outputs.
184     */
185     pwm01 = PWM_in1;
186     pwm02 = PWM_in2;
187     pwm03 = PWM_in3;
```

The Output window shows the following build results:

```
Build Find in Files
Copyright (c) 2003 Mic:
Errors : 0
MP2COD 3.40, COFF to O
Copyright (c) 2003 Mic:
Errors : 0
MP2HEX 3.40, COFF to H
Copyright (c) 2003 Mic:
Errors : 0
Loaded C:\mcc18\Projec
BUILD SUCCEEDED: Mon O
```



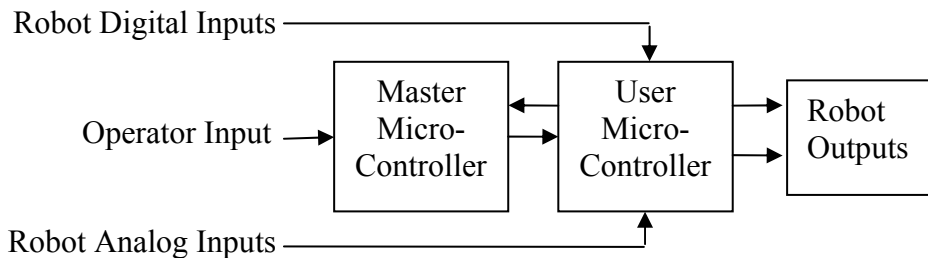
Table of Contents

1. PICmicro Processor and Memory	3
1.1. Processors	3
1.2. PICmicro MCU Program Memory	3
1.3. Default vs. Custom Programs	3
1.4. Programming Language and Windows Software	4
2. PICmicro Software Structure	4
2.1. Commenting Format	6
2.2. Declare Variables	6
2.3. Define Aliases	7
2.4. Define Constants	7
2.5. Initialization	7
2.6. Main Loop call: Process_Data_From_Master_uP();	8
2.7. Main Loop - Getdata Data Input	8
2.8. Main Loop – Default_Routine call: Perform Operations Here	9
2.9. Main Loop - Putdata Data Output	10
2.10. Main Loop – Return to main.c	10
3. Program Inputs	11
3.1. Digital Inputs	12
3.2. Analog Inputs	13
4. Program Outputs	15
4.1. PWM Analog Outputs	15
4.2. Standard Digital Outputs	17
4.3. Relay Digital Outputs (FRC only)	17
4.4. Solenoid Digital Outputs (EDU only)	18
5. Downloading User Code to the Robot Controller	19
5.1. Downloading Steps	19
5.2. PROGRAM STATE LED	19
6. Additional Programming Techniques	20
6.1. Serial Debug Statements	20
6.2. Robot Feedback and User Mode (FRC RC only)	21
6.3. RC Mode – Competition, Autonomous, and User (FRC RC only)	23
7. PICmicro Commands	24

1. PICmicro Processor and Memory

1.1. Processors

Each Robot Controller has two built-in PICmicro 18F8520 microcontrollers from Microchip Technology. One of these PICmicros uses memory that is programmable by the user. This programmability allows customization of user controls, semi-automatic robot functions, and autonomous robot functions. This User processor has access to all operator input data and robot sensor data, as well as control over the PWM and digital outputs used to control motors, relays, and solenoids.



1.2. PICmicro MCU Program Memory

The PICmicro in each Robot Controller has onboard FLASH memory that can be used to store the user's custom programs. This memory is non-volatile; meaning that it keeps its memory after power has been removed. There is a total of 32k bytes of memory space available to the user in which to store custom programs.

The program in the Robot Controller can be changed by downloading a new program into the PICmicro memory through the PROGRAM port (more details on page 16).

1.3. Default vs. Custom Programs

The Robot Controller is supplied with a "Default" program loaded into memory in order to help get the robot up and running quickly. When more sophisticated control of the robot is desired, a custom program can be quickly created by modifying the Default program. Default "Source Code" for the default programs are provided at www.InnovationFirst.com. Since Innovation First offers both a smaller EDU Robot Controller and a full-size Robot Controller, two different sets of code are provided on the web site. The programs are not interchangeable from one type to the other, but the manner of programming is the same.

1.4. Programming Language and Windows Software

Programs for the Robot Controller can be written in Assembly or C. For ease of use, the default code has been written in the C programming language. This high-level language was selected because it is very powerful and has become an industry standard. Should the user desire, Assembly code can also be mixed with the C code.

Throughout this document it is assumed that the reader is familiar with the C programming language and its conventions.

MPLAB IDE is the name of the free Windows program used to edit, compile, and debug PICmicro programs. *MPLAB C18* is the name of the C compiler that must also be installed for use with *MPLAB IDE*. Microchip has developed a special version of this C compiler for use with Innovation First Robot Controllers. This product, called C-BOT, is included in some Robot Controller kits, or it may be purchased separately from Innovation First.

For downloading custom programs to the Robot Controller, a program named *IFI Loader* must be used. This is included in the C-BOT package, or you can get this program as well as the Default Code for your Robot Controller as free downloads from www.InnovationFirst.com.

2. PICmicro Software Structure

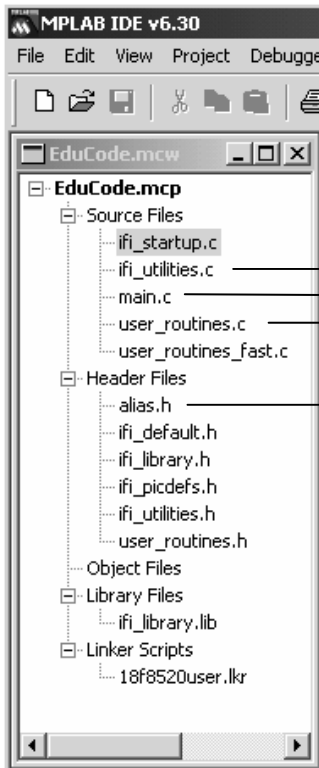
Inside *MPLAB IDE* is where the Robot Controller's program is edited and compiled. From the "File/Open Workspace..." menu you should open the workspace (.mcw) file downloaded from the Innovation First web site. You will then see a window listing all the files used in the project (.mcp). All of these files are necessary, but only a few of them should be edited by the user to create a custom program. See the next page for details on which files should be edited. To open a file for viewing or editing, double click on it in the workspace window.

The PICmicro program used in the Innovation First Robot Controllers must adhere to a pre-defined structure. The structure does not limit the capability of your software; the structure ensures that the Robot Controller operates properly. This structure is described in the block diagram below.

You can refer to Microchip's *MPLAB C18 C Compiler Libraries* manual for more information on PICmicro commands. That document will not teach you to program in C, but it details libraries specific to the PICmicro architecture. Not all of the libraries can be used to program the Robot Controller, as specific functions and registers are reserved for correct operation in the Robot Controller system.

For Assembly language instructions and detailed PICmicro device information, refer to the 18FXX20 datasheet from Microchip.

Note that there is a 64 character path/name limit inherent in the Microchip compiler. That means that you will get a compilation error if you try to compile a project which has any file whose path and name length exceeds 64 characters.



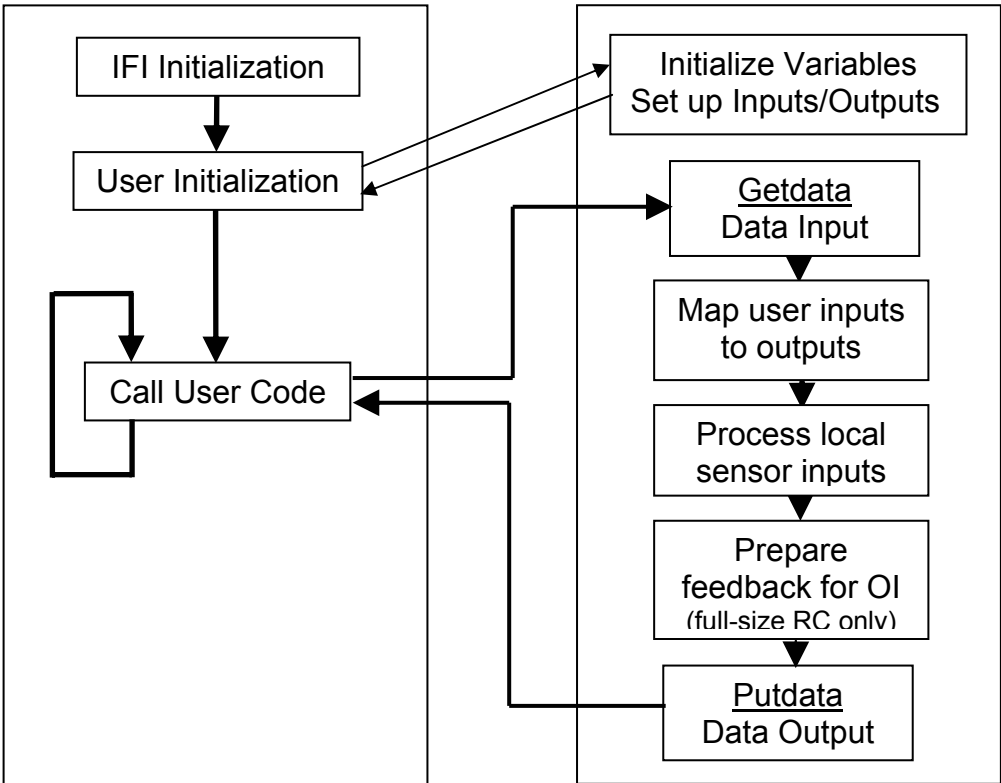
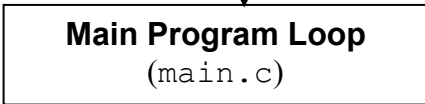
Handy functions that the user can call. User can add to this file.

All other files MUST NOT be changed!

User may add custom aliases in this file.

No need to modify this file.

Edit this file



2.1. Commenting Format

Commenting means that the text is for informational purposes to the reader only and is not processed by the microcontroller. Lines are commented by enclosing the text within the following structure: `/* */`

Comment Line Examples: The line of code below is completely commented (line starts with `/*` and ends with `*/`) and is used for informational purposes. It has no effect on the operation of a program.

```
/* This text is for informational purposes and is not processed by the compiler. */
```

Comments can span more than one line:

```
/* Begin comment  
   Continue comment  
   End comment  
*/
```

You cannot nest comment structures. The following is illegal and will not compile:

```
/* This is /* not allowed */ and won't work. */
```

2.2. Declare Variables

All variables used in the programs must be declared by type and name. The 18F8520 has 2kb of data memory available in which to store variables. Variables may be declared in the `.c` file where they are used. There are two primary types of variables that are used in the Robot Controller code. For variables whose values might range from 0 to 255, the type of `unsigned char` is used. When values might range from 0 to 65535, `unsigned int` is used. All other types within C are also possible, including `double` and `float` (32-bit) types.

The default code defines all the standard input and output variables. The names of the variables used in the default program can be changed; however, use caution since the name is used throughout the default code and must be changed at every instance. Declare any additional variables required.

Declare Variables Examples:

```
/* This is how variables are declared. */  
unsigned char    loop_count;  
unsigned int     output_value;          /* Don't forget the semicolon! */  
  
/* You can declare more than one variable of the same type per line. */  
unsigned char    loop_count1, loop_count2, loop_count3;  
  
/* You can also initialize the value of variables at the same time you declare them. */  
unsigned int     countdown = 10;  
float            pi = 3.1415;
```

Sharing Variables Between Files Example: If you want to use the same variable (and its value) in more than one file, this is how you would do it. This is commonly referred to as a global variable.

```
unsigned int wheel_turns = 25;          /* Declared in user_routines.c, for example. */  
  
extern unsigned int wheel_turn;        /* Put at the top of user_routines_fast.c, for example.*/
```

2.3. Define Aliases

Aliases provide alternate and sometimes shorter names for variables and sub-divisions of variables. Aliases don't require any additional memory. The default code uses Aliases to provide logical names for the Analog Inputs, Digital Inputs, Relay Outputs, Solenoid Outputs, and PWM Outputs. Aliases can also be used to point to a specific bit within a byte, as in the example below. In the default code, all aliases have been placed in the `alias.h` file. You may add your own aliases to this file, but do not delete any existing ones.

Define Aliases Example: Each line declares one alias. After each variable you can place comments on the use of the alias in your application. You can see how aliases can save you a lot of typing.

```
#define countup      countdown      /* countdown and countup refer to the same variable. */
#define pl_sw_trig  rxddata.oi_swA_byte.bitselect.bit0 /*Joystick Trigger Button */
#define rc_dig_int01 PORTBbits.RB2 /* Refers to a specific input pin of the PICmicro. */
#define rc_dig_in01 rc_dig_int01 /* Multiple aliases for the same item are allowed. */
```

2.4. Define Constants

Constants are fixed values with a name assigned to them. Constants do not require any memory or code space. Constants provide a convenient means to organize, locate, and edit fixed values in the code.

Define Constants Example: Each line below assigns one constant. The Default Code has several examples of constants.

```
#define DATA_SIZE      30
#define SPI_TRAILER_SIZE 2
#define SPI_XFER_SIZE   DATA_SIZE + SPI_TRAILER_SIZE
#define PWM_NEUTRAL     0x7F /* This is how to write the hexadecimal number 7F. */
```

2.5. Initialization

Initialization is called from the `main.c` file. First a routine called `IFI_Initialization()` is called, which sets up crucial system parameters. This call should not be deleted or tampered with.

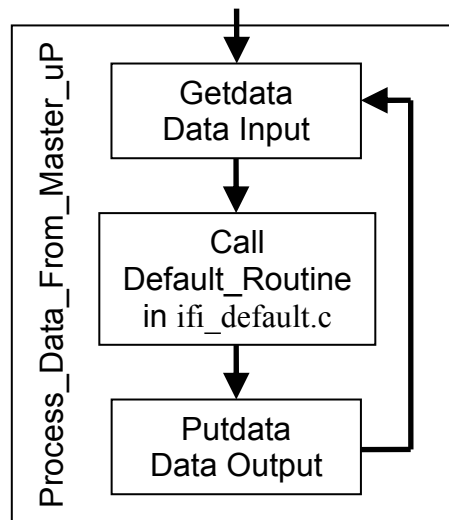
Next the routine called `User_Initialization()` is called. This function resides in the `user_routines.c` file and may be edited by the user. In this routine the multi-function pins of the Robot Controller can be set up as either digital inputs or digital outputs (or analog inputs for the EDU Robot Controller). Follow the examples given in the code to accomplish this.

Not all pins, registers, functions, and peripherals of the 18F8520 User processor are available to the user. In the initialization section, as throughout your custom code, be careful not to address, use, or otherwise modify any of them which are already in use elsewhere in the default code. For a list of features which are off-limits or unusable, please look in the `ifi_picdefs.h` file. Ports and registers which should not be used are marked with the comment: `/* Reserved - Do not use */`

2.6. Main Loop call: `Process_Data_From_Master_uP();`

The core function of the default PICmicro program is to call the `Process_Data_From_Master_uP()` function. This routine performs the following:

1. Read In the Input Data from the operator and on-board robot sensors
2. Perform specific functions based on the Input Data
3. Send Out the results to perform physical movement of the robot's motors and valves.



This loop structure ensures that 1) new inputs (joysticks, etc.) are continuously read in, and 2) the outputs (motors, Victors, Spikes, etc.) continuously receive new commands. This function is called at a fixed rate of 26ms for the FRC Robot Controller and 17ms for the EDU Robot Controller.

While this is the maximum rate at which new data is received from the remote operator, it is possible to execute faster code by placing it outside the fixed timing loop, inside the `Process_Data_From_Local_IO()` routine in the `user_routines_fast.c` file.

2.7. Main Loop - Getdata Data Input

The first part of the Main Loop is the Getdata command. This command reads a serial stream of data from the Master processor. For proper operation of the Robot Controller this command should not be moved or modified.

2.8. Main Loop – Default_Routine call: Perform Operations Here

After the Getdata function is called and new data is received, the Default_Routine() routine is called. This routine is also located in user_routines.c and this is where all the default operation of the robot is assigned. You can modify this routine to re-map the inputs to outputs, and to perform special functions.

Perform Operations Example: The main loop below is just an example. Refer to the Default Code for a more complete example. The code below performs the following functions:

1. Port 1 Y axis linked to PWM1. PWM2 and PWM3 are set to neutral.
2. Port 3 Trigger activates Relay 3 in the Forward direction. See the Relay Outputs on page 17.
3. Port 3 Thumb button activates Relay 3 in the Reverse direction.
4. Robot digital input 1 automatically activates the roller forward at the speed set by constant ROLLERSPEED.

```
/*===== PERFORM OPERATIONS =====*/
/*----- Joystick to PWM -----*/
pwm01 = p1_y;
pwm02 = 127;          /* Set to neutral. */
pwm03 = PWM_NEUTRAL; /* Another way to set to neutral. Remember our constant? */

/*----- Button to Relay -----*/
relay3_fwd = p3_sw_trig; /* Relay 3 Forward = Port 3 Trigger */
relay3_rev = p3_sw_top;  /* Relay 3 Reverse = Port 3 Thumb button */

                          /* relay3_fwd is an Alias to a bit an output pin. */
                          /* p3_sw_trig is an Alias to a bit in oi_swA_byte. */

/*----- Roller Code -----*/
if (rc_dig_in01 == 0)    /* Don't forget to use a double-equals for conditionals! */
{
    Roller_Out = ROLLER_SPEED;
}
else
{
    Roller_Out = PWM_NEUTRAL;
}
```

2.9. Main Loop - Putdata Data Output

The final part of the Main Loop is the Putdata command. This command sends a serial stream of data back to the Master processor. The Master processor then uses this data to drive some of the PWM outputs. Do not delete, modify, or move the Putdata command line. If it is not executed at least every 26.2ms, the Master processor will disable the User processor and your RC will not function. Therefore you should not put any infinite loops in your code, or any other code which might take a very long time to execute.

It is recommended that you set unused PWM outputs to 127, unused relay and digital outputs to 0, and unused solenoid outputs to 0.

On the FRC, the Master processor will disable all outputs if there is no communication with the Operator Interface or if the Competition Mode is Disabled, unless you set the Robot Controller to run in Autonomous mode.

2.10. Main Loop – Return to main.c

The Main Loop is now complete. Program execution will return to the `while(1)` in the `main.c` file and this loop will repeat forever.

3. Program Inputs

The following sections describe the types of data input to the PICmicro program. The User program receives some input data from the Master processor inside the Robot Controller. The other inputs come from local sensors and ports on the Robot Controller which are fed directly to the User processor.

For a FIRST Robotics Control System which uses an RS-422 Radio for communicating with an Operator Interface, the human operator's controls are passed into the Robot Controller through the radio port. The Master processor handles this communication and then sends the control signals to the User processor.

For an EDU Robot Controller the human operator's commands are sent using a standard hobby radio-control (R/C) transmitter and receiver pair. The hobby R/C receiver is interfaced with the Robot Controller as detailed in the Robot Controller's user guide. The Master processor takes the signals from the hobby R/C receiver and digitizes them before sending them to the User processor.

The next two sections will talk about the two types of inputs that are received by the User processor inside the Robot Controller.

3.1. Digital Inputs

Digital Inputs are defined as inputs that are either HIGH or LOW. Buttons and switches are the most common form of digital inputs. In software a digital input is either a zero (0 = LOW) or a one (1 = HIGH). The Digital Inputs on the Robot Controllers and Operator Interfaces have pull-ups, which means that they are normally high, or one. That means that if a digital input is open or not connected, the value read by the PICmicro digital input is HIGH (=1). If a digital input is connected to ground, the input is LOW (=0). The buttons on the joysticks are an example of digital inputs to the Operator Interface.

Operator Interface Digital Inputs

Only the FRC Robot Controller can communicate with the Operator Interface. Refer to the Operator Interface Reference Manual for details on the Digital Inputs. The sixteen (16) Digital Inputs from the Operator Interface are grouped into two bytes, OI_SWA and OI_SWB. These input variables are aliased into more meaningful names such as p1_sw_trig which refers to the trigger of the Joystick connected to Port 1.

Digital Inputs: The Operator Interface Digital Input aliases are found in the file `alias.h` of the Default Code.

```
/*----- Example aliases for each OI switch input -----*/
#define p4_sw_trig  rxdata.oi_swB_byte.bitselect.bit4 /*Joystick Trigger Button*/
#define p4_sw_top   rxdata.oi_swB_byte.bitselect.bit5 /*Joystick Top Button*/
#define p4_sw_aux1  rxdata.oi_swB_byte.bitselect.bit6 /*Aux input*/
#define p4_sw_aux2  rxdata.oi_swB_byte.bitselect.bit7 /*Aux input*/
```

Robot Controller Digital Inputs

Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Digital I/O pins. They are pins which go directly to the User processor. These pins can be configured as either inputs or outputs in your initialization.

The Default Code already has some of these I/O pins set up as Inputs that you can use with no change to the initialization code. They have aliases as in the following examples.

Digital Inputs: The Robot Controller Digital Input aliases are found in the file `alias.h` of the Default Code.

```
/*----- Example aliases for default FRC RC digital inputs -----*/
#define rc_dig_in13  PORTGbits.RG4
#define rc_dig_in14  PORTCbits.RC0
#define rc_dig_in15  PORTJbits.RJ4
#define rc_dig_in16  PORTJbits.RJ5

/*----- Example aliases for default EDU RC digital inputs -----*/
#define rc_dig_in13  PORTHbits.RH4
#define rc_dig_in14  PORTHbits.RH5
#define rc_dig_in15  PORTHbits.RH6
#define rc_dig_in16  PORTHbits.RH7
```

NOTE: There are differences between the pin-outs of each type of Robot Controller. This is why the full-size and EDU Robot Controllers each have a different version of code and they are not interchangeable.

3.2. Analog Inputs

Analog Inputs are defined as inputs that can have any voltage value between 0 and 5 volts, not just ON or OFF. Joysticks, Potentiometers, and Gyro sensors are typically connected to Analog Inputs. Both the Robot Controller and the Operator Interface (FRC system only) have the capability to measure analog inputs.

Operator Interface Analog Inputs

Refer to the Operator Interface Reference Manual for details on the Analog Inputs. The sixteen (16) Analog Inputs from the Operator Interface are each stored in bytes, such as `p1_x` and `p1_y` corresponding to the X and Y Joystick axis connected to Port 1. The CH FlightStick joystick used with the Operator Interface has three analog inputs per joystick, the X axis, the Y axis, and the Wheel.

In software, an analog input from the Operator Interface (FRC only) is represented as a number from 0 to 254. The table below shows how an Analog Input varies as the joystick is moved. In the FRC Robot Controller's Default Code, the Analog Inputs from the Operator Interface are mapped directly to the PWM Outputs without modification (see PWM Outputs on page 15). The EDU system uses the joysticks on standard hobby radio controllers as remote analog inputs. These inputs are similarly mapped directly to the PWM Outputs on the Robot Controller.

Joystick Function	Position and Analog Input Value		
Y Axis	Full Forward = 254	Neutral = 127	Full Back = 0 – 25*
X Axis	Full Left = 254	Neutral = 127	Full Right = 0 – 25*
Wheel	Full Forward = 254	Neutral = 127	Full Back = 0 – 25*

* Note: The joystick axes rarely go all the way to zero. This is normal. See the PWM section on page 15.

Analog Inputs: The Operator Interface Analog Input aliases are found in the file `alias.h` of the Default Code.

```

/*----- Operator Interface (OI) - Analog Inputs -----*/
#define p1_y      rxdata.oi_analog01
#define p2_y      rxdata.oi_analog02
#define p3_y      rxdata.oi_analog03
#define p4_y      rxdata.oi_analog04
#define p1_x      rxdata.oi_analog05
#define p2_x      rxdata.oi_analog06
#define p3_x      rxdata.oi_analog07
#define p4_x      rxdata.oi_analog08
#define p1_wheel  rxdata.oi_analog09
#define p2_wheel  rxdata.oi_analog10
#define p3_wheel  rxdata.oi_analog11
#define p4_wheel  rxdata.oi_analog12

```

Radio Control PWM Analog Inputs

Refer to the EDU System Reference Manual for details on interfacing it with standard hobby radio control (R/C) transmitters and receivers. The joystick movements from the transmitters then become inputs to the Robot Controller, with their corresponding values stored in bytes.

Analog Inputs: The hobby R/C receiver input aliases are found in the file `alias.h` of the Default Code.

```
/*----- Aliases for each radio-control receiver PWM input -----*/
#define PWM_in1      rxdata.oi_analog01
#define PWM_in2      rxdata.oi_analog02
#define PWM_in3      rxdata.oi_analog03
#define PWM_in4      rxdata.oi_analog04
```

Robot Controller Analog Inputs

The Robot Controller Analog Inputs have a 10-bit resolution. Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Analog Inputs. (Note: The Analog Inputs of the EDU Robot Controller must be configured in the initialization section, as they can also be used as Digital I/O if desired.) The Default Code already has some of these Analog Inputs configured so that you can use them with no change to the initialization code. The FRC Robot Controller has all 16 analog inputs configured by default. The EDU Robot Controller has only the first four configured.

In order to use them you must call the `Get_Analog_Value` function, which resides in the `ifi_utilities.c` file. This function will return a 10-bit value stored in an unsigned `int`, or two bytes. See the example below if you want to assign it to a 8-bit variable, like a PWM output.

Analog Inputs: The Robot Controller Analog Inputs are found in the file `alias.h` and used as follows.

```
/*----- Robot Controller (RC) - Analog Input Aliases -----*/
#define rc_ana_in01  ADC_CH0
#define rc_ana_in02  ADC_CH1
#define rc_ana_in03  ADC_CH2
#define rc_ana_in04  ADC_CH3

/* Example of using the Analog Inputs to map an analog input to a PWM output. */

unsigned int sensor1;
unsigned char sensor1_8bits;

sensor1 = Get_Analog_Value( rc_ana_in01 );      /* Assign the analog reading to a variable. */
sensor1_8bits = (unsigned char)(sensor1 >> 2); /* Only take the 8 most significant bits, */
pwm01 = sensor1_8bits;                          /* because PWM OUTPUTS can only be 8 bits. */

/* Or you can do it all in one line like this. */
pwm01 = (unsigned char) (Get_Analog_Value(rc_ana_in01) >> 2);
```

4. Program Outputs

The following sections describe the types of data and actions that are output from the PICmicro program. Most of the output pins of the Robot Controller are directly tied to the User processor and can be addressed by the user in their code.

4.1. PWM Outputs

The PWM Output Ports on the Robot Controllers are all controlled by PICmicro software. Each PWM output provides variable speed control for a motor in both Forward and Reverse.

PWM stands for Pulse Width Modulation and is the type of signal commonly used to control Speed Controllers such as the Victor 884. This PWM signal is the standard hobby R/C type used on model airplane controls. This PWM is not the regular duty-cycle type output by most microcontrollers.

PWM Outputs: The PWM outputs have aliases defined in the `alias.h` file. By assigning a value from 0 – 254 to them you can control the speed of a motor.

```
/*----- Example Aliases for Robot Controller PWM outputs -----*/
#define pwm01      txdata.rc_pwm01
#define pwm02      txdata.rc_pwm02
#define pwm03      txdata.rc_pwm03
#define pwm04      txdata.rc_pwm04
```

In the EDU system any given PWM output signal can be created by either the Master or the User processors. See the `Setup_Who_Controls_Pwms` function in the `User_Initialization` routine of the `user_routines.c` file in the EDU Default Code to do this. In the Default Code the Master processor controls all PWMs according to the values stored in `pwm0x` variables and sent in the `PutData()` function. This means that PWM outputs can only be changed every 17ms when the Master gets the new values from the User processor. If the User processor is set up to control one or more PWM outputs, it can change those outputs every program loop if desired by using the `Generate_Pwms` function.

EDU Robot Controller PWM Outputs: Here are some example lines of code to show how the PWMs are controlled.

```
/* Setup Master processor to control PWMs 1-4, User to control PWMs 5-8. */
Setup_Who_Controls_Pwms(MASTER,MASTER,MASTER,MASTER,USER,USER,USER,USER);

pwm01 = pwm02 = pwm03 = pwm04 = pwm05 = pwm06 = pwm07 = pwm08 = 42;

/* Every 17ms Master will generate PWMs 1-4 according to values stored in pwm0x variables. */
Putdata(&txdata);

/* The User processor will generate PWMs 5-8 with whatever values are passed as arguments to
this function. Notice that the values for PWMs 1-4 will be ignored, since the Master
controls them. Also notice that PWM 8 will be set at neutral. It must be type-cast. */
Generate_Pwms(1,Z,ZERO,ZERO,pwm05,pwm06,pwm07,(unsigned char)127;
```

In the FRC system the values for PWM signals 1-16 are all sent to the Master processor from the User processor using the `PutData()` function. The Master processor, however, only generates the signals for PWMs 1-12. The signals for PWMs 13-16, however, are generated directly by the User processor by using the `Generate_Pwms` function. As with the EDU RC, this means that PWMs 1-12 can only be changed every 17ms, while 13-16 can be changed as often as the program loop executes. You must use both the `Putdata(&txdata)` and the `Generate_Pwms(pwm13, pwm14, pwm15, pwm16)` commands to generate all 16 PWM output signals.

The most common method of controlling a PWM output is via the joystick Analog Input. By sending a joystick Analog Input to a PWM output, a motor connected to the PWM port will act as follows:

Action (FRC system)	Function and Analog Value		
PWM Speed Controller	Full Forward = 254-227	Neutral = 136-123	Full Reverse = 0 – 37
Joystick Y Axis	Full Forward = 254	Neutral = 127	Full Back = 0 – 25
Joystick X Axis	Full Left = 254	Neutral = 127	Full Right = 0 – 25
Joystick Wheel	Full Forward = 254	Neutral = 127	Full Back = 0 – 25

Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the PWM Outputs.

Example (using the default FRC software): Move the Y-axis on the Port 1 Joystick forward to make a motor connected to PWM1 turn. The further you push the joystick, the faster the motor turns. From a neutral position move the Y-axis on the Port 1 Joystick backwards to make a motor connected to PWM1 turn the other direction. Again, the further you push the joystick, the faster the motor turns.

FRC Robot Controller PWM Outputs: The FRC default code maps the joystick Analog Inputs from the Operator Interface to the PWM outputs as follows. Elsewhere in the code, the `Generate_Pwms()` and `Putdata()` functions are called to generate the PWM output signals.

```

/*----- Analog Inputs (Joysticks) to PWM Outputs-----
*-----
*   This maps the joystick axes to specific PWM outputs.
*/
pwm01 = p1_y;
pwm02 = p2_y;
pwm03 = p3_y;
pwm04 = p4_y;
pwm05 = p1_x;
pwm06 = p2_x;
pwm07 = p3_x;
pwm08 = p4_x;
pwm09 = p1_wheel;
pwm10 = p2_wheel;
pwm11 = p3_wheel;
pwm12 = p4_wheel;

Generate_Pwms(pwm13, pwm14, pwm15, pwm16);
Putdata(&txdata); /* Send my data to the master microprocessor. */

```


4.2. Standard Digital Outputs

Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Digital I/O pins. They are pins which go directly to the User processor. These pins can be configured as either inputs or outputs in your initialization.

The Default Code already has some of these I/O pins set up as Outputs that you can use with no change to the initialization code. They have aliases as in the following examples.

Digital Outputs: The Robot Controller Digital Output aliases are found in the file `alias.h` of the Default Code.

```

/*----- Example aliases for default FRC RC digital outputs -----*/
#define rc_dig_out13    LATGbits.LATG4
#define rc_dig_out14    LATCbits.LATC0
#define rc_dig_out15    LATJbits.LATJ4
#define rc_dig_out16    LATJbits.LATJ5

/*----- Example aliases for default EDU RC digital outputs -----*/
#define rc_dig_out13    LATHbits.LATH4
#define rc_dig_out14    LATHbits.LATH5
#define rc_dig_out15    LATHbits.LATH6
#define rc_dig_out16    LATHbits.LATH7

/*----- Example usage of digital outputs -----*/
if (pwm01 > 227)          /* If pwm01 is greater than 227... */
{
    rc_dig_out1 = 1;      /* then output a HIGH on digital output port 1 */
    rc_dig_out2 = rc_dig_in3; /* and map digital input 3 to digital output 2. */
}

```

NOTE: There are differences between the pin-outs of each type of Robot Controller. This is why the full-size and EDU Robot Controllers each have a different version of code and they are not interchangeable.

4.3. Relay Outputs (FRC only)

The Relay ports on the FRC Robot Controller are all controlled by PICmicro software. The Relay outputs provide Full Forward, Full Reverse, and OFF control for motors and other devices when connected with a Spike Relay. Each relay port has 2 digital outputs. Refer to the appropriate Robot Controller Reference Manual for details on the number and location of the Relay Outputs.

Example (using the default software): Press the Trigger button on the Port 1 Joystick to put a Spike Relay connected to RLY1 in the Forward position. This would cause a motor connected to this relay to turn at its maximum speed. Press the Top button on the Port 1 Joystick to put the Spike in Reverse position, thereby reversing the motor. The relays in the example below are limited by limit switches connected to Digital Inputs 1 and 2.

Relay Digital Outputs: The relay outputs have aliases defined in the `alias.h` file. Here is an example of how to use them.

```

/*----- Example Aliases for Robot Controller relay outputs -----*/
#define relay1_fwd    LATDbits.LATD0
#define relay1_rev    LATEbits.LATE0
#define relay2_fwd    LATDbits.LATD1
#define relay2_rev    LATEbits.LATE1

```

```

/*----- Example usage of relay ports -----*/
/* When the switch on rc_dig_in01 is closed, the value will be 0. */
/* This will make relay1_fwd also be 0, no matter what the value of p1_sw_trig is. */
  relay1_fwd = p1_sw_trig & rc_dig_in01;
  relay1_rev = p1_sw_top & rc_dig_in02;

```

4.4. Solenoid Digital Outputs (EDU only)

The solenoid outputs on the EDU Robot Controller are essentially treated as digital outputs in the software. A high, or 1, will energize (turn on) the solenoids, and a low, or 0, will shut them off.

Example (using the default software): Push the joystick corresponding to R/C PWM IN 3 forward past the programmed threshold of 160 to energize solenoid 1. Pull it backwards past 95 to energize solenoid 2. In the neutral position both solenoids will be off.

Solenoid Digital Outputs: The solenoid outputs have aliases defined in the `alias.h` file. Here is an example of how to use them.

```

/*----- Example Aliases for EDU Robot Controller solenoid outputs -----*/
#define solenoid1      LATDbits.LATD0
#define solenoid2      LATDbits.LATD1
#define solenoid3      LATDbits.LATD2
#define solenoid4      LATDbits.LATD3
#define solenoid5      LATDbits.LATD4
#define solenoid6      LATDbits.LATD5

/*----- R/C PWM INPUTs toggle solenoids -----*/

  if (PWM_in3 < 95)          /* PWM_in3 forward (typically) */
  {
    solenoid1 = 1;          /* turns on Solenoid 1 */
    solenoid2 = 0;          /* turns off Solenoid 2 */
  }
  else if (PWM_in3 > 160)    /* PWM_in3 reverse (typically) */
  {
    solenoid1 = 0;          /* turns off Solenoid 1 */
    solenoid2 = 1;          /* turns on Solenoid 2 */
  }
  else                       /* PWM_in3 neutral band */
  {
    solenoid1 = 0;          /* turns off Solenoid 1 */
    solenoid2 = 0;          /* turns off Solenoid 2 */
  }

```

5. Downloading User Code to the Robot Controller

5.1. Downloading Steps

The Robot Controller downloading steps are as follows:

1. Compile a valid program in *MPLAB IDE* by selecting “Make” from the “Project” menu. This will generate a `.hex` file with the same name as the project name. You may also use the `xxx_Default.hex` file which has been pre-compiled to restore the default functionality.
2. Power ON the Robot Controller.
3. Connect a DB9 Male-to-Female Pin-to-Pin cable (maximum length 6 ft.) from the PROGRAM port on the Robot Controller to PC’s serial port.
4. Run the *IFI Loader* application from Start -> Programs -> IFI_Loader -> IFI_Loader.
5. Right click and select the appropriate COM port in the lower right corner of the *IFI Loader* window.
6. Click the “Browse” button and select the desired `.hex` file which you compiled from within *MPLAB IDE*.
7. (EDU RC only) Briefly press the “PROG” button on the Robot until the Program State LED becomes solid orange. Now it is in Program Download mode. (On some PCs, this step must also be performed for the FRC Robot Controller by pressing the “Download” button.)
8. Click the “DOWNLOAD” button in *IFI Loader*. The Program State LED will blink orange.
9. When the download is complete the Robot Controller will automatically reset. There is no need to power cycle it. If the program is good, the Program State LED will blink green. A terminal window will automatically open within *IFI Loader* where any text output from the Robot Controller will be displayed.

Note: Before the new User program will start executing, an FRC Robot Controller must have a Valid RX (link) to an OI or the RC must be in Autonomous Mode.

5.2. PROGRAM STATE LED

Depending on the status of the User processor, the PROGRAM STATE LED will be green, orange, or red, and blink or stay on solid.

After programming the Robot Controller you should see the PROGRAM STATE LED flashing green. This LED indicates that the new program is running. For the FRC system you must also be linked to an Operator Interface or else this LED will be solid green and the program will not be running.

The PROGRAM STATE LED will be a solid orange if it is waiting for new user code to be downloaded. This will occur if the PROG button is briefly pressed on the Robot Controller. Pressing this button is only necessary for the EDU Robot Controller. It may also be necessary for the FRC Robot Controller on some PCs. If you are unable to download to the FRC Robot Controller, you may need to press the PROGRAM button to change the LED to orange.

While a new program is being downloaded, the PROGRAM STATE LED will blink orange.

The PROGRAM STATE LED will be red if the Master processor is unable to communicate with the User processor. This can occur if something in the User Code interferes with the inter-processor communication by writing to a reserved area.

6. Additional Programming Techniques

If you leave the serial cable connected to your Robot Controller after programming it, the following routines can be used to print data to your computer screen. You can display the data in any serial communications terminal application. *IFI Loader* has a Terminal Window which can be used for this purpose, or you may use the application HyperTerminal. HyperTerminal is located in Start->Programs->Accessories->Communications. Settings for HyperTerminal are: Bits per second: 115200, Data bits: 8, Parity: None, Stop bits: 1, Flow control: None. You might also want to check "Append line feeds to incoming line ends" in "File->Properties->Settings->ASCII Setup...".

These commands provide a convenient way for your PICmicro program to send data to the PC screen while running. Some possible uses include debugging programs by showing you the value of a variable or expression, or by indicating what portion of a program is currently executing. Once your program has been debugged, however, you should remove all such serial text output commands so that your code will run as fast as possible in your robot.

6.1. The printf() Command

The most common method to print text output to a screen in C is the printf() command. While this command is not included in the Microchip C library, IFI has implemented a limited model of printf in the `printf_lib.c` file. Please examine this file to learn the limitations of the IFI implementation as well as to see examples of how to create your own custom output modules. The default code has the following example included to get you started.

printf() Example: The following example demonstrates the use of a printf() command to display the decimal values of PWM OUT 7 and 9 to the computer screen.

```
printf("PWM07 = %d, PWM08 = %d\n", (int)pwm07, (int)pwm08); /* printf example */
```

Notice that the values displayed (`pwm07` and `pwm08`) are of type unsigned char and must be typecast as int to work with this command.

6.2. Other Serial Debug Statements

While the printf() function is a standard C library, its use requires more processing power than the following utilities. The commands which follow are faster methods of printing data to a PC terminal screen, although they do not allow the flexibility and formatting of a printf statement. These commands are contained in the file `ifi_utilities.h` and can be useful for debugging your programs when execution speed is important.

You may also create your own custom serial output functions using these as models.

PrintByte Example: The following example demonstrates using the `PrintByte` command to print bytes of data in hexadecimal format to the computer screen.

```
/*----- Somewhere in the Main Loop -----*/
PrintByte(0xC4); /* Print a hard-coded value. */
PrintByte(pwm01); /* Print a variable value. */
```

After you add these lines somewhere in the main loop, and after downloading, the *IFI Loader* will open a Terminal Window on your PC screen and wait for data from the Robot Controller. You should see “C4” and “7F” repeating, each on their own lines. “7F” should vary depending on the value of `pwm01`, an output variable. These lines will repeat every loop of the code. Note that if you close the Terminal Window, your program keeps executing, but you can't see the debug data anymore. You can reopen Terminal Window from the Options menu. Optionally, you can use HyperTerminal to view the data.

PrintWord Example: This function will print a word of data, or two bytes, in hexadecimal format.

```
/*----- Somewhere in the Main Loop -----*/
PrintWord(0xC4A1); /* Print a hard-coded value. */
```

You should see “C4A1” printed on your screen. A carriage return is automatically added.

PrintString Example: You can also print text strings using this function.

```
/*----- Somewhere in the Main Loop -----*/
PrintString( (unsigned char *) "ERROR1\r" ); /* Print a text string. */
```

This example will display “ERROR1”. The “\r” performs a carriage return. To use this function, simply replace “ERROR1” with the text you would like to print.

DisplayBufR Example: This function will display all 26 bytes of either the TXDATA or the RXDATA buffer. In this manner you can snoop on the data being sent to or received from the Master processor. It is a good candidate for customization so that you can display your own long buffers.

```
/*----- Somewhere in the Main Loop -----*/
DisplayBufR( &rxdata ); /* Print the 26 bytes received from the Master processor. */
DisplayBufR( &txdata ); /* Print the 26 bytes sent to the Master processor. */
```

This example will dump all 26 received bytes on one line before inserting a carriage return and then printing the 26 transmitted bytes with another carriage return. See what happens if you use this and move the joysticks on your control system!

6.3. Robot Feedback and User Mode (FRC only)

The FRC Robot Controller has the ability to send data back to the Operator Interface. The Robot Controller can send 11 bits every loop of the code. The data is available on the Operator Interface in three ways: 1) On the 11 Robot Feedback LEDs, 2) on the multi-segment display when in user mode, and 3) on a PC connected to the Dashboard Port. See the `PB_Mode` section (page 23) for writing code that can display numbers on the multi-segment display.

The Robot Feedback bits are passed from the User processor to the Master processor in the txdata packet. The Master processor then sends the data to the Operator interface via the Radio Modems. You can change the Feedback bits one at a time as in the example below.

Robot Feedback Example: The Default Code assigns aliases to the bits which control the Robot Feedback LEDs on the Operator Interface. These aliases are defined in the `alias.h` file. The user can change these aliases.

```
/*----- Example aliases for the ROBOT FEEDBACK LEDs -----*/
#define pwm1_green    txdata.LED_byte1.bitselect.bit0
#define pwm1_red      txdata.LED_byte1.bitselect.bit1
#define pwm2_green    txdata.LED_byte1.bitselect.bit2
#define pwm2_red      txdata.LED_byte1.bitselect.bit3
```

The values of these LEDs (ON or OFF) are set in the Default Code in the `ifi_default.c` file. Here is an excerpt.

```
if (p1_y >= 0 && p1_y <= 56)
{
    /*Joystick is in full reverse position*/
    pwm1_green = 0;    /*Turn PWM1 green LED - OFF */
    pwm1_red   = 1;    /*Turn PWM1 red LED   - ON   */
}
```

6.4. RC Mode – Competition, Autonomous, and User (FRC RC only)

The `rc_mode_byte` is a byte accessible through the GetData command. The Competition Mode, Autonomous Mode, and User Mode are all available in this byte. The Default Code already contains all the code needed to use these modes.

Competition Mode

Bit 7 of `rc_mode_byte` (aliased as `competition_mode` below) indicates the status of the Competition Control, either Enabled or Disabled. This indicates the starting and stopping of rounds at the competitions. Competition Mode is indicated by a solid "Disabled" LED on the Operator Interface. When Disabled, all program execution is stopped.

`competition_mode` = 1 for Enabled, 0 for Disabled.

Autonomous Mode

Bit 6 of `rc_mode_byte` (aliased as `autonomous_mode` below) indicates the status of the Autonomous Mode, either Autonomous or Normal. This indicates when the robot must run on its own programming. When in Autonomous Mode, all OI analog inputs are set to 127 and all OI switch inputs are set to 0 (zero). Autonomous Mode is indicated by a blinking "Disabled" LED on the Operator Interface.

`autonomous_mode` = 1 for Autonomous, 0 for Normal.

Autonomous Mode can also be turned ON by setting the RC to Team 0 (zero) and power cycling the RC unit. Note that if `competition_mode` = 0 (Disabled), then Autonomous Mode is also disabled.

User Mode

Bit 5 of `rc_mode_byte` (aliased as `user_display_mode` below) indicates when the user selects the "User Mode" on the OI. `PB_mode.bit5` is set to 1 in "User Mode". When the user selects channel, team number, or voltage, `user_display_mode` is set to 0. When in "User Mode", Robot Feedback LEDs are turned off and the value stored in `txdata.LED_byte1.data` is displayed on the multi-segment display on the Operator Interface. In the default code this is set to the value of Port 1 Y-Axis.

Note: "User Mode" is identified by the letter "u" in the left digit (for 4 digit OI's)

Note: "User Mode" is identified by decimal places on the right two digits (for 3 digit OI's)

Autonomous Mode Example: This example shows portions of the code that reference `rc_mode_byte`. The Perform Operations Section below uses aliases based on `rc_mode_byte` bits to execute different sections of code.

The following three aliases are declared at the end of the `alias.h` file:

```
#define user_display_mode rxdata.rc_mode_byte.mode.user_display
#define autonomous_mode rxdata.rc_mode_byte.mode.autonomous
#define competition_mode rxdata.rc_mode_byte.mode.disabled
```

For code you wish to have run in Autonomous mode, you must edit the `User_Autonomous_Code()` routine in the `user_routines_fast.c` file. Here is an excerpt from `main.c` which calls the autonomous code:

```
if (autonomous_mode)          /* DO NOT CHANGE! */
{
    User_Autonomous_Code();    /* You edit this in user_routines_fast.c */
}
```

User Mode Example: This example shows the Default Code that changes the data sent to the Operator Interface when the user sets the Operator Interface to User Mode.

```
if (user_display_mode == 0) /*User Mode is Off */
{
    ...
    /* This section of code deleted for this example. */
    ...
} /* (user_display_mode = 0) (User Mode is Off) */

else /* User Mode is On */
{
    txdata.LED_byte1.data = p1_y; /* Change p1_y to a value in which you are interested. */
}
```

7. PICmicro Commands

Not all features of the 18F8520 PICmicro can be used by the user who is writing custom code for the Innovation First Control System. Certain registers are reserved because they are used to allow the User processor to function within the embedded system. Modification of these registers may result in improper operation or in the Robot Controller failing to function at all.

Refer to the file `ifi_picdefs.h` to find out which registers and bits are reserved by the system, or are not available to the user. It is not necessarily comprehensive, since many registers have multiple functions, some of which may be allowed, while others are not. For a complete pin list of the usable pins which you can access in your custom code, please refer to the appropriate Robot Controller user guide.

If your Robot Controller ceases functioning after a program change, you should revert back to the last known working version and then incrementally add back in the new code until you find what caused the problems. It is good programming practice to keep an archive of old code so that you can do this.