Renard PLUS Lighting Control

Pic Programming Guide

Oct 2013

Doc Rev 0.01

Contents

1. Introduction to Renard Pic Programming

You are likely reading this because you just finished building your first of many Renard light controllers. You realized that beyond the PCB, parts, soldering, casing, and wiring of the hardware, there is something called "Programming" that is needed. You might have also realized it is all a bit confusing and you need some guidance. Congratulations, you have opted to research the process and try to learn about it instead of just getting frustrated and getting mad at the developers. Hopefully, you have come to the right place and we can help educate, guide, and generally support you in your efforts. Before we get into the software, bit twiddling, setting Vpp, flashing, erasing, etc. we need a little background.

Renard is a lighting controller concept developed by Phil Short. He developed a set of hardware and firmware (the software) to have a microcontroller (originally the "PIC" 16F688 available from MicroChip*) respond to a computer generated serial command/data stream he defined as "Renard Protocol". To teach the PIC how to read and respond appropriately to "Renard" he developed the original Renard PIC source code. This, with a tool-chain/IDE known as MPLab (free from MicroChip), can be compiled into a binary image (HEX file) that is then programmed into the PIC. If you have a "canned" (pre-configured) HEX file, the process to program it is fairly straight forward. However, if you need to customize the code in any way, then things get a bit more complex as we will try to explain.

Another thing to keep in mind is that you need to obtain the proper code (either source or HEX files) for the appropriate hardware. Since the time Mr. Short developed Renard, there have been many innovators like Mac, Dean, Brian, etc. who have created designs and enhanced Renard to fill out the designs available to address user's desired capabilities. The Renard-Plus line, including the "Simple" series and "Plus" series of boards is a great example of this. These controllers have adapted new features, or used different versions of the PIC microcontroller than the original 16F688 part Mr. Short originally used. That means different code is necessary to run Renard on those boards. If you have an SS16, you need the code indicated for the SS16, if you have an RP32, you need the RP32 code (either source or canned HEX) FOR the RP32, etc. Some code covers a number of different designs. For the Renard Plus controllers, obtaining the proper code is easy because the code, (source and canned HEX), is available on the appropriate product page on the www.renard-plus.com website. There are also other "flavors" of Renard code, such as the Renard DMX available for some boards that allows a Renard board to learn a different communications protocol (see Wikipedia on DMX).

We will try to address all of this in this document and try to make the options a little less confusing.

*Other brands and names may be claimed by others

2. What do I need?

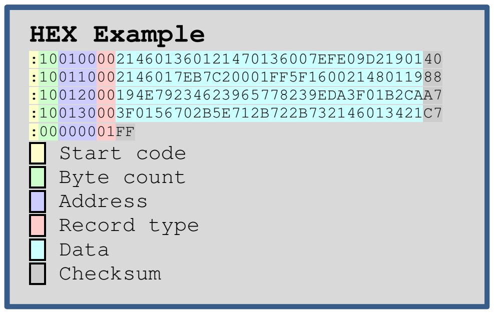
That is a good question, but to answer it we need to understand a few more details before we can answer. What you need depends on the board you are using and what you want to do with the code. For example, if your requirements match an existing HEX file configured the way you need, and you have a Renard-Plus design, the list of requirements to be able to program is much shorter. Other situations can be more complicated. For our examples, we will mostly talk about the Renard Plus boards as we know them best. The general information applies to programming any of the currently known Renard controllers so if you are an SS, Renbus, Headblinker, or any of the other Renard variants, this information should help you also.

What I am doing	List of needs	From where
Have a precompiled HEX for my board that matches my needs.	The hex file	The web (e.g. renard-plus.com). email, a WIKI, where ever.
	PIC Programmer[1]	(see description)
	Programmer S/W e.g. PICKit	Microchip, or Pic Programmer vendor
I have source because my use is not typical.	Renard Source (appropriate to board)	The web (e.g. renard-plus.com)
	MPLabs	MicroChip website
	Some programming experience	Various places including MicroChip website.
	PIC Programmer[1]	(see description)
	Programmer S/W e.g. MPLabs	Microchip, or Pic Programmer vendor

[1]NOTE: If you have a board that does NOT support on-board programming through an ICSP connector, then you likely will additionally need a ZIF socket if your programmer does not include one – see the ZIF Sockets section for more details.

2.1. What is HEX

HEX is a file format for a firmware data developed by Intel for handling binary data without the difficulties associated with the potential issues with raw binary data files. It represents binary code in an ascii text format that is easier to handle, easier to transfer electronically, and safer to transport. It incorporates checksums so data integrity of a file can be verified. It allows addressing of the data so it can comprehend gaps in the binary data it represents without having to "pad" the file with unnecessary blank data. Most importantly, it is the file format required for most PIC programmers, and is the output format of the compiled source that MPLabs* outputs. See the Wikipedia here: http://en.wikipedia.org/wiki/Intel_HEX


HEX Example

```
:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011988
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

▢	Start code
▢	Byte count
▢	Address
▢	Record type
▢	Data
▢	Checksum

for more details. Many Renard boards offer "pre-complied" HEX files for "canned" situations. These typically mean you are using common serial interface data rates, typical clock settings, typical features, etc. Again, it is important to match the HEX file to your board requirements. For Renard Plus, if you get a HEX file from the product page for your board type, then you have the right thing, and for FREE! For other boards, you may have to dig, or go to the board developer (if they are still around) to try to get code. The Wiki on many of the do-it-yourself Christmas lighting forums are a good place to start your search. You can also ask forum members (like at diychristmas.org) to compile a HEX for you (MOST members are friendly and helpful) if you are not able to handle those steps. You might be able to get a HEX file, or you might get access to source to compile and if the latter is the case, you need to compile it as described in the MPLabs section later in this guide before you will have a HEX to program.

2.2. Firmware Options

The Renard hardware is designed to be very flexible and as such, there are a variety of options in the software to exploit that flexibility. Here we will discuss the more common firmware options that you may want or need to use.

2.2.1. Start Address

The Renard protocol is designed to daisy-chain controllers in the order you want them addressed. If you have one controller that has 16 channels first in line then a controller with 8 then one with 32- the data stream from the PC first goes into the first controller which takes the first 16 channels for itself and passes on the rest of the data stream (minus the first 16 channels) to the next controller. That next controller takes the first 8 channels of data, and passes along the rest (minus the first 8). The next controller takes the first 32 channels and passes the rest. Etc. Every controller in this situation takes the first data off the stream (whatever number of channels it needs) and passes along the rest. From the PC, it looks like the first controller gets channels 1-16, the next 17-24, the next 25-56, etc. This is great if you can order your display in a daisy chain fashion, and you are not using wireless which are two reasons you might need to set a start address.
As you may have noticed, in the example we talked about board channel counts in blocks of 8 addresses. The is because Renard started based on multiples of 16F688 PICs that can control 8 channels and the software evolved around the concept that a Renard will consume blocks of 8 channels. The start address is designed the same way- it is calculated such that a start address tells you how many blocks of 8 channes the controller will ignore BEFORE it responds to the data stream. Thus if you set a start address of 01, that means it will ignore 1x8=8 channels of the data stream before starting to respond therefore starting at address 9. A 02 means 2x8=16 so it ignores the first 16 channels in the data stream effectively starting at channel 17. And so on. To calculate the actual start address the calculation is (nx8)+1 = start address where "n" is the start address number. We cannot help you plan out your addresses, you will need to layout your yard and figure out what will need to map where.

2.2.2. Baud Rate

Baud rate is the speed in bits per second at which the communications between the computer and controller are run. A faster baud rate means more data in less time which could mean faster display updates and more channels supported. The primary way to communicate with a Renard is via a serial COM port (that uses either RS232 or RS485 depending on your adapter/port). Most operating systems (like Windows) have a maximum baud rate supported is usually either 115K or 128K with 57.5K (57600) as no problem. The plugin for Vixen 2.x also tends to top out at 57600 but other animation software might support higher. Renard controllers in general support a max of

57600 but Renard Plus controllers go higher to 115K which is the practical top speed without running into stability issues. So basically, you need to pick the top speed possible with your hardware/software and use that value for your setting. With all Renard Plus and latest animation software, you should be able to support 115k baud but it might be wise to double check your software to see if it supports it. On the Renard Plus site under the product pages, you will find archives of both 57600 and 115K re-complied. With source, you can set it to what you want to support.

2.2.3. PWM vs non-PWM

PWM stands for Pulse Width Modification. This is a mechanism that the PIC uses to do dimming. Basically the PIC can pulse the control off and on at a rate that allows the lights to dim (assuming your lights support this) by having the power lowered. If you have a situation where you want the controller to be 100% on whenever the board is told the lights should be on (anywhere from 1% to 100% dim level) then you disable PWM in the code to accomplish this. Non-dimable devices include florescent lights (the non-dimable kind), motors, servos, light strings with their own control box, etc. The usual is to have PWM enabled so you can dim "normal" light strings including most DC lights.

2.2.4. Zero Cross

Zero Cross refers to a hardware mechanism used to determine when the AC line power used to power lights is crossing the 0 volt level of the alternating current. Remember AC swings + and – using a sine wave and 120 time a second (for 60 hz) it hits 0 instantaneous volts as it starts to swing the current the other direction. The controllers need to know when this happens so they can effectively PWM the SSR triac output. A triac (used in most SSR) latches on and stays on until the AC voltage drops to 0V then it can unlatch (unless the control signal tells it to stay on). If we want to DIM a triac based SSR, we watch for zero cross, then hold off enabling the triac until a period of time AFTER zero cross occurs thus limiting the voltage/current going out and achieving dimming. You may want to disable Zero Cross detect in some designs if a line AC signal is not available or you needed to disable AC dimming.

2.2.5. Renard vs Renard DMX?

One decision you need to make is if you will use Renard protocol, or DMX on your Renard. This is not something we can tell you to use one or the other – we can just discuss the pros and cons. Please note that for some controllers the DMX version is a totally separate source base from the Renard code, and in some cases it is a build option in the source code.

2.2.5.1. Renard Protocol

The Renard Protocol was originally developed by Phil Short as THE code to use on a Renard and has been exclusive to the do-it-yourself lighting community. It is well supported by a loyal subset of do-it-yourselfers and enjoys a robust development following so innovations and improvements are being made to it all the time. It is the most common firmware that is run on the Renard hardware so getting help with it is very easy on the do-it-yourself lighting forums. It is based on a simple serial data stream (that can be RS232 or RS485 if you need distance) so interfaces are widely available, however, the handling and transmission of the data stream is a software thing so there are limits on how many different serial connections (for large "universe" displays) can be handled without stuttering or dropping packets. The standard protocol is designed to have a controller delete the data it uses from the stream and pass along the rest. This makes addressing boards EASY (they are all 0) as the location on the serial cable determines the board order. This does not work well if you need to scatter around your channel address in your yard (such as 1-8 on one side and 9-16 on the other then back to 17-??) which can end up with a mess of data cables when changes in the physical board order are needed. Wireless is also a problem so to solve this there is the ability to force a start address (on an 8 channel boundary) if needed for Renard for limited cases. Maximum data rate can also be an issue as Renard is limited to the lower data rates of a typical serial port – 57.6K is typical and 115K (maybe higher) is possible with some serial ports with tweaks to the OS and/or animation software.

2.2.5.2. DMX on Renard

DMX, actually DMX512 and the related SACN E1.31 protocol, is a commercial lighting industry standard and has a large following in the rest of the light animation world. See http://en.wikipedia.org/wiki/DMX512 for details. If you buy a commercial lighting product, it will likely be DMX protocol that it understands. Also DMX is used by other commercial lighting systems like LOR, etc. so if you need to mix other stuff in with your Renard, it might be a good idea to use DMX on your Renard. It has a much higher data rate capability as the standard calls out a 250K baud data rate. To achieve that, DMX usually requires a special DMX interface that handles the transmission of the data stream in hardware so you are less likely to run into the troubles of feeding the data stream via software. However, those interfaces are less common than RS232 and RS485 so these are much more expensive to obtain. DMX exclusively uses start addressing, so the serial stream is the same everywhere rather than being modified along the way by each controller. This is good for wireless although achieving wireless rates at DMX speeds has been an issue in the past and just recently solved with the new NRF transceiver based solutions now available.

2.2.5.3. Renard vs DMX Summary

Both protocols are viable with the nod going to Renard for a pure do-it-yourself environment and cheaper interfaces. DMX is good for mixing in commercial products WITH the do-it-yourself gear but has a higher entry price because of the interface. We leave the choice to you although a board can usually be easily reprogrammed back and forth so you could start with one and switch to the other if you need. Yay choice!!!

2.3. Picking a Pic Programmer (HW/SW)

The next thing you need is the actual programmer. A programmer is usually a specialized piece of hardware that can manipulate the non-volatile (stays with power off) memory in a particular device. It knows how to handle the erase/program/verify/read steps to do that memory manipulation. For the PIC parts, there are a number of very good options available to you for purchase from a variety of sources. Again, yay choice!! Which one you pick depends on what you need to do vs price/performance. Here are some details that will be helpful to know:

2.3.1. PICKit

Microchip, so far, has developed a number of programming and debugger solutions for their PIC micros. The most popular, by far, are the PICKit2* and PICKit3*. Both have the ability to program a PIC separately (in an add-on ZIF socket), OR, if the Renard board offers an ICSP port, like Renard Plus, you can do programming "on board".

Microchip offers the top of the line PICKit3: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en538340&redirects=pickit3

*Other brands and names may be claimed by others

However, MicroChip has shared the design of the PICKits and allows other compatible/identical "third party" programmers to be offered usually at a lower price than the MicroChip ones! (Hey, it is good for the PIC developer world to have choices). Because of that, there are many good "third party" or aftermarket or "clone" PICKit programmers available to you. You can purchase from places like Mouser, Alibaba, EBay, DIY Lighting stores, etc. depending on your risk vs reward level and the price vs quality ratio you are willing to endure.

ICSP Connector
30cm
USB
iCP02v2
- PIC10F,12F,16F,18F,24F, dsPIC30,33
- EEPROM 11LCxx, 24LCxx, 25LCxx, 93LCxx
- Compatible with PICkit2, MPLAB IDE
- Windows XP, Vista, 7, Linux, Mac OS (32/64)

There are some great inexpensive options, but there is also some real junk. For beginners it is best to start with a PICKit or compatible and use a vendor, like the ones that support the DIY Lighting community. You can also look on Ebay, or Alibaba if you want- it is YOUR choice. But, you might ask, what about PICKit2 vs 3? We try to answer that NEXT so read on!

2.3.1.1. PICKit 2 vs 3

The MicroChip site PICKit 3 manual (http://ww1.microchip.com/downloads/en/DeviceDoc/PICkit_3_User_Guide_51795A.pdf) has this to say:

> The PICkit 3 programmer/debugger system is similar in function to the PICkit 2 in-circuit debugger system. Similarities of the two debuggers include:
> - Powered via USB cable to PC
> - Provides a programmable voltage power supply
>
> The PICkit 3 differs from the PICkit 2 by providing:
> - Extended EE program image space (512 Kbytes)
> - True voltage reference
> - Increased voltage range (1.8-5V VDD; 1.8-14V VPP)

Generally this means that newer parts, like PIC32 or future chips MIGHT only be supported by a Pickit3 or compatible. It is important to note that there are no known plans of changing the Renard designs to use any of the higher end parts that PICKit 2 cannot program so this capability is not very important for blinky use. Anecdotal discussions on the web (Try Googling "PICKit2 vs 3") seem to say that the PICKit 3 is slower and less reliable than PICKit 2 and has fewer features. For example, the PICKit 2 has a capability of doing a 3 channel logic analyzer that does not seem to be available on PICKit 3. The author, while documenting the PICKit 3 software, ran into some difficulties running it on Windows 7 x64 but fortunately found running it in "Admin Mode" (Google it) seemed to correct most of its bad behavior.

The bottom line is that EITHER version of the PICKit (or compatible) is a good choice for blinky work. In the following section, we provide programming instructions for both flavors of the PICKit software.

2.3.2. PICKit software

So you have picked your PICKit and have a hex file you want to program. What now? Well here we walk you through an example programming process for the PICKit 3 (the PICKit 2 is identical operation by design).

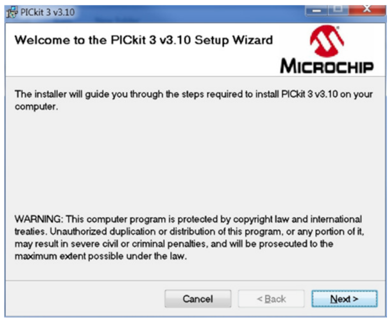
2.3.2.1. PICKit 3 Programming Example

Let's assume you built a Renard Plus TR16 and just received your PICKit 3. Here is what you do:

1. Use the included CD of software, OR Download and Install your PICKit 3 software from MicroChip:

http://ww1.microchip.com/downloads/en/DeviceDoc/PICkit3%20Programmer%20Application%20v3.10.zip

Unzip and read the ReadMe.txt file for details on installing. For this version you need to unzip "PICkit3 Programmer Application Setup v3.10.zip" and then run Setp.exe to install.



Welcome to the PICkit 3 v3.10 Setup Wizard

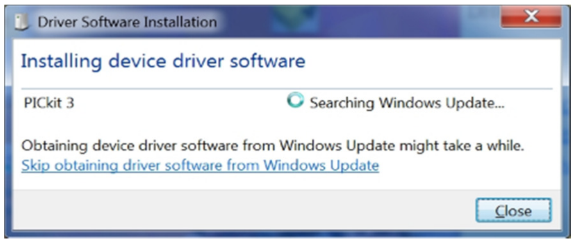
MICROCHIP

The installer will guide you through the steps required to install PICkit 3 v3.10 on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Generally, just select the default options and everything should work. You may need to install additional .NET pieces for the program to work but this is normal. **If you are running Win7 or higher, you will need to set the shortcut attributes to run in Admin Mode or it will not work.**

2. Plug in your PICkit 3. First time you should see dialog about driver installation. This is an example for Win7:

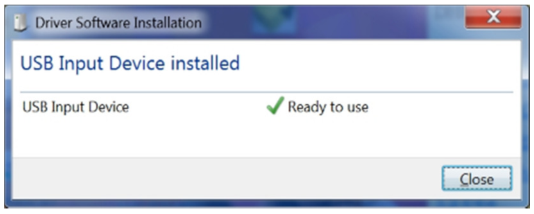


Installing device driver software

PICkit 3 — Searching Windows Update...

Obtaining device driver software from Windows Update might take a while.

Skip obtaining driver software from Windows Update



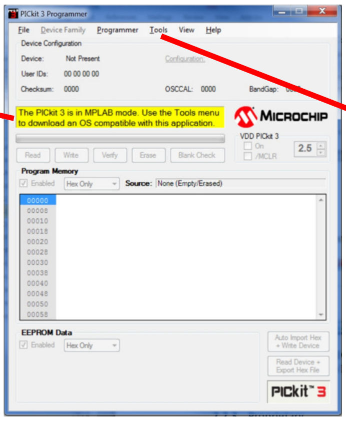
USB Input Device installed

USB Input Device — ✓ Ready to use

3. Now unplug the programmer from the PC, attach it to the board (with part installed) in the ICSP port (or the ZIF socket with the part installed in it) and plug the PICKit back into the PC. Leave the board powered off- the PICKit provides the power to program. Watch pin 1 orientation!!

4. Now run the PICKit 3 v3.10 program from the icon on your desktop or from the Windows Start/Programs/Microchip/PICKit 3 v3.10:

Note: special instructions to follow first time for the programmer. You may not need to do this step if your PICKit is loaded with the correct firmware. See PICKit software guide for more details.

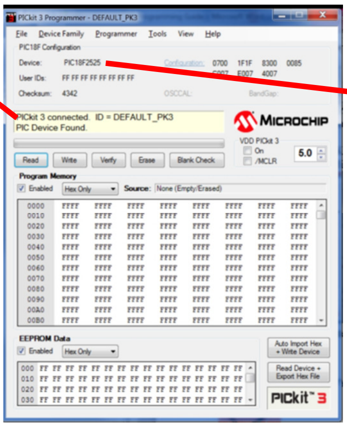

PICkit 3 Programmer
File Device Family Programmer Tools View Help
Read Write Verify Erase Blank Check
Auto Import Hex + Write Device
Read Device + Export Hex File
PICkit 3

```
Device Configuration
Device:    Not Present                Configuration:
User IDs:  00 00 00 00
Checksum:  0000            OSCCAL:  0000       BandGap:

The PICkit 3 is in MPLAB mode. Use the Tools menu
to download an OS compatible with this application.

VDD PICkit 3   On  /MCLR   2.5
Program Memory   Enabled   Hex Only   Source: None (Empty/Erased)
EEPROM Data      Enabled   Hex Only
```

Tools button to correct warning

After the firmware update, upon startup of the PICKit 3 while attached to a board ICSP (or ZIF with target part in the socket):

Successful PIC detection.


PICkit 3 Programmer - DEFAULT_PK3
File Device Family Programmer Tools View Help
Read Write Verify Erase Blank Check
Auto Import Hex + Write Device
Read Device + Export Hex File
PICkit 3

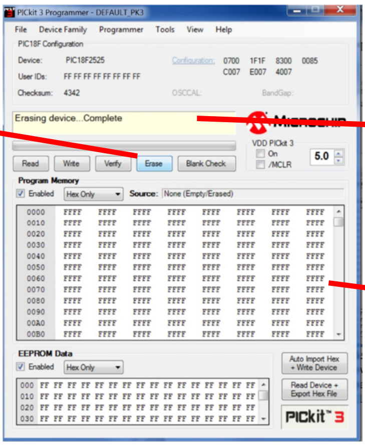
```
PIC18F Configuration
Device:    PIC18F2525        Configuration:  0700  1F1F  8300  0085
                                             C007  E007  4007
User IDs:  FF FF FF FF FF FF FF FF
Checksum:  4342              OSCCAL:         BandGap:

PICkit 3 connected.  ID = DEFAULT_PK3
PIC Device Found.

VDD PICkit 3   On  /MCLR   5.0
Program Memory   Enabled   Hex Only   Source: None (Empty/Erased)
```

PIC type detected.

This is what things should look like when it is all running properly and we are ready to start the programming process. Note that the programmer found the PIC18F2525 that the Renard Plus TR16 uses in the example we are showing.

5. Before any other steps, we need to ERASE the part:

Erase Button


PICkit 3 Programmer - DEFAULT_PK3
File Device Family Programmer Tools View Help
Read Write Verify Erase Blank Check
Auto Import Hex + Write Device
Read Device + Export Hex File
PICkit 3

```
PIC18F Configuration
Device:    PIC18F2525        Configuration:  0700  1F1F  8300  0085
                                             C007  E007  4007
User IDs:  FF FF FF FF FF FF FF FF
Checksum:  4342              OSCCAL:         BandGap:

Erasing device...Complete

VDD PICkit 3   On  /MCLR   5.0
Program Memory   Enabled   Hex Only   Source: None (Empty/Erased)
0000  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0010  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0020  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0030  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0040  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0050  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0060  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0070  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0080  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
0090  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00A0  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
00B0  FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
EEPROM Data      Enabled   Hex Only
000 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

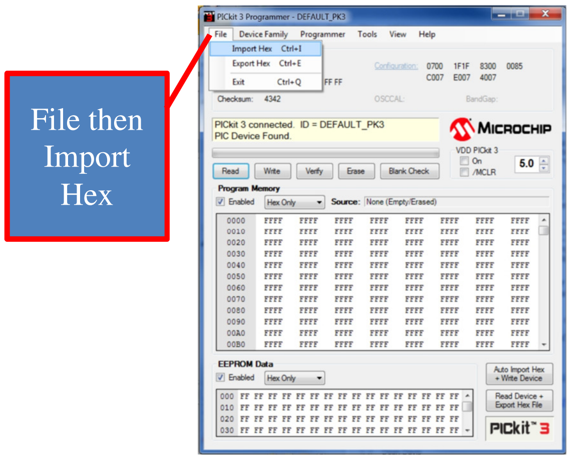
Successful erase of the PIC

Note all FFFF blank data!

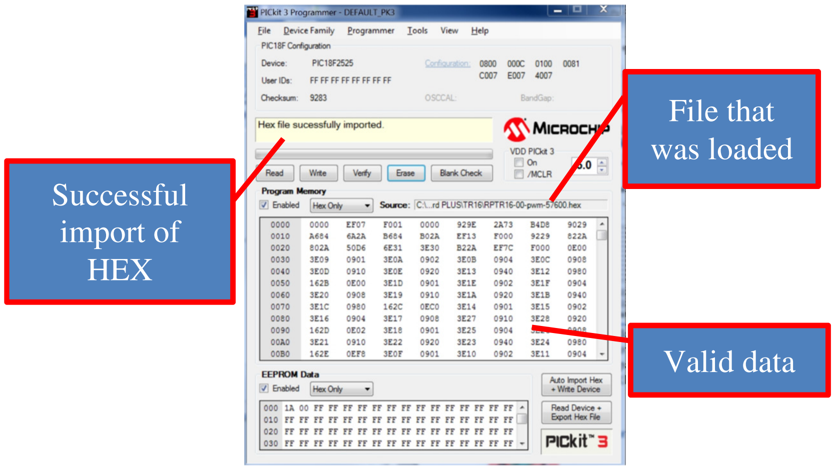
If you load the HEX file, then trigger an erase, it will blank the data in memory you just loaded so ERASE before importing HEX files and keep in mind you should double check the Program Memory window.

*Other brands and names may be claimed by others

6. Next we need to load the precompiled HEX file (this step is always AFTER erasing). You should have already downloaded from the http://renard-plus.com site on the product page for your board or from the appropriate repository for your particular board. For this example, we have accessed the TR16 product page and picked the 57600 baud rate package and downloaded. After opening the ZIP package, we extracted the 00 address file (to remain compatible with other Renards on the chain) so the example file name is RPTR16-00-57600.hex (you need to pick the appropriate file for your situation and keep it where you remember where to access it.)

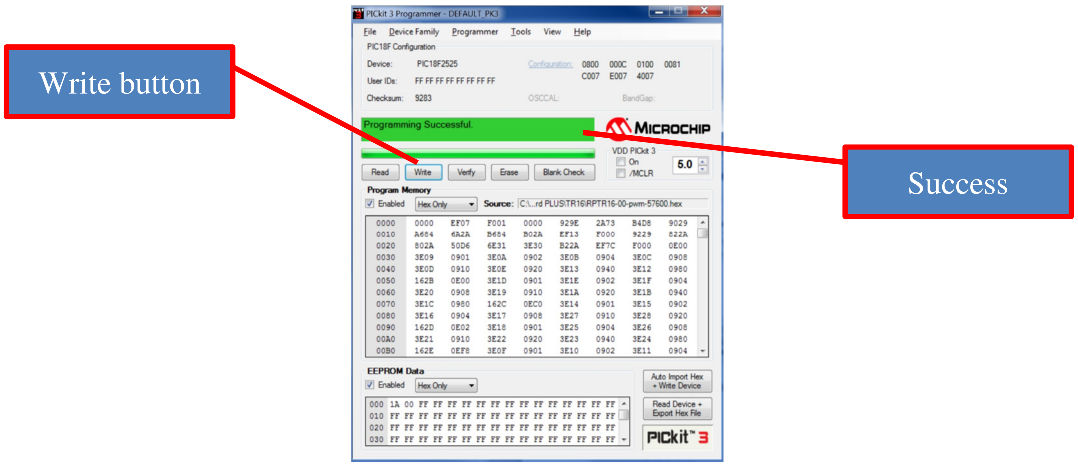


```
Configuration:  0700  1F1F  8300  0085
                C007  E007  4007
FF FF
Checksum:   4342          OSCCAL:          BandGap:

PICkit 3 connected.  ID = DEFAULT_PK3
PIC Device Found.

VDD 5.0

Program Memory
[✓] Enabled   Hex Only   Source: None (Empty/Erased)

0000  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0010  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0020  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0030  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0040  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0050  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0060  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0070  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0080  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
0090  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
00A0  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF
00B0  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF  FFFF

EEPROM Data
[✓] Enabled   Hex Only

000  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```



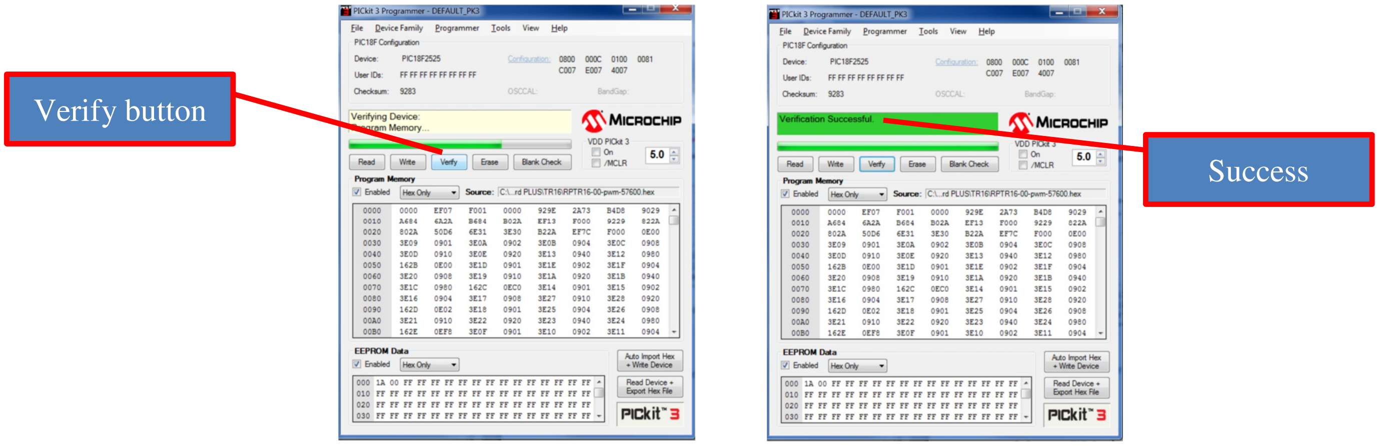
```
PIC18F Configuration
Device:     PIC18F2525    Configuration:  0800  000C  0100  0081
                                          C007  E007  4007
User IDs:   FF FF FF FF FF FF FF FF
Checksum:   9283          OSCCAL:          BandGap:

Hex file sucessfully imported.

Program Memory
[✓] Enabled   Hex Only   Source: C:\...rd PLUS\TR16\RPTR16-00-pwm-57600.hex

0000  0000  EF07  F001  0000  929E  2A73  B4D8  9029
0010  A684  6A2A  B684  B02A  EF13  F000  9229  822A
0020  802A  50D6  6E31  3E30  B22A  EF7C  F000  0E00
0030  3E09  0901  3E0A  0902  3E0B  0904  3E0C  0908
0040  3E0D  0910  3E0E  0920  3E13  0940  3E12  0980
0050  162B  0E00  3E1D  0901  3E1E  0902  3E1F  0904
0060  3E20  0908  3E19  0910  3E1A  0920  3E1B  0940
0070  3E1C  0980  162C  0EC0  3E14  0901  3E15  0902
0080  3E16  0904  3E17  0908  3E27  0910  3E28  0920
0090  162D  0E02  3E18  0901  3E25  0904  3E26  0908
00A0  3E21  0910  3E22  0920  3E23  0940  3E24  0980
00B0  162E  0EF8  3E0F  0901  3E10  0902  3E11  0904

EEPROM Data
[✓] Enabled   Hex Only

000  1A 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

7. Now we program by clicking the "Write" button. You will see status messages roll by in the status window and if the programming completes you will see:



```
PIC18F Configuration
Device:     PIC18F2525    Configuration:  0800  000C  0100  0081
                                          C007  E007  4007
User IDs:   FF FF FF FF FF FF FF FF
Checksum:   9283          OSCCAL:          BandGap:

Programming Successful.

VDD 5.0

Program Memory
[✓] Enabled   Hex Only   Source: C:\...rd PLUS\TR16\RPTR16-00-pwm-57600.hex

0000  0000  EF07  F001  0000  929E  2A73  B4D8  9029
0010  A684  6A2A  B684  B02A  EF13  F000  9229  822A
0020  802A  50D6  6E31  3E30  B22A  EF7C  F000  0E00
0030  3E09  0901  3E0A  0902  3E0B  0904  3E0C  0908
0040  3E0D  0910  3E0E  0920  3E13  0940  3E12  0980
0050  162B  0E00  3E1D  0901  3E1E  0902  3E1F  0904
0060  3E20  0908  3E19  0910  3E1A  0920  3E1B  0940
0070  3E1C  0980  162C  0EC0  3E14  0901  3E15  0902
0080  3E16  0904  3E17  0908  3E27  0910  3E28  0920
0090  162D  0E02  3E18  0901  3E25  0904  3E26  0908
00A0  3E21  0910  3E22  0920  3E23  0940  3E24  0980
00B0  162E  0EF8  3E0F  0901  3E10  0902  3E11  0904

EEPROM Data
[✓] Enabled   Hex Only

000  1A 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
020  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

8. Not absolutely required but just a darn good idea and highly recommended is doing a Verify after the write. This will read back the part memory and compare against what should have been programmed. It is rare to have a verify fail, but it does happen!


Verify button
Success
File Device Family Programmer Tools View Help
Read Write Verify Erase Blank Check
Auto Import Hex + Write Device
Read Device + Export Hex File
PICkit™ 3

```
PICkit 3 Programmer - DEFAULT_PK3
PIC18F Configuration
Device:    PIC18F2525          Configuration:  0800  000C  0100  0081
                                               C007  E007  4007
User IDs:  FF FF FF FF FF FF FF FF
Checksum:  9283                OSCCAL:              BandGap:

Verifying Device:
Program Memory...

VDD PICkit 3   [ ] On   [ ] /MCLR   5.0

Program Memory
[✓] Enabled   Hex Only   Source: C:\...rd PLUS\TR16\RPTR16-00-pwm-57600.hex
0000  0000  EF07  F001  0000  929E  2A73  B4D8  9029
0010  A684  6A2A  B684  B02A  EF13  F000  9229  822A
0020  802A  50D6  6E31  3E30  B22A  EF7C  F000  0E00
0030  3E09  0901  3E0A  0902  3E0B  0904  3E0C  0908
0040  3E0D  0910  3E0E  0920  3E13  0940  3E12  0980
0050  162B  0E00  3E1D  0901  3E1E  0902  3E1F  0904
0060  3E20  0908  3E19  0910  3E1A  0920  3E1B  0940
0070  3E1C  0980  162C  0EC0  3E14  0901  3E15  0902
0080  3E16  0904  3E17  0908  3E27  0910  3E28  0920
0090  162D  0E02  3E18  0901  3E25  0904  3E26  0908
00A0  3E21  0910  3E22  0920  3E23  0940  3E24  0980
00B0  162E  0EF8  3E0F  0901  3E10  0902  3E11  0904

EEPROM Data
[✓] Enabled   Hex Only
000 1A 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF
010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
020 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

```
PICkit 3 Programmer - DEFAULT_PK3
Verification Successful.
```

Once it verifies, your firmware is on the chip and you can disconnect the programmer (or remove it from the ZIF socket and install back in the board) and fire up the board to test.

Common Problems:

- Verifies but does not work in good board: You might have forgotten to reload the HEX file after erasing. The program tends to clear out its buffer on an erase so you must reload after every erase. This is an easy one to get messed up on- it happens to the author all the time. A hint is to look for data in the Program Memory window on the PICKit software before programming. If you see all FFFF, then you probably do not have a HEX file loaded.

- Fails to verify: If you forget to erase a part that was previously programmed and then program with new data, you can get a program succeeded, but fails to verify. You simply need to start over and follow the programming process remembering to erase first, then load.

- Fails to program: This may be a bad chip or an issue with the programming socket (i.e. the PIC socket on a Renard Plus board). Things to try- check the board for unsoldered pins on the PIC or ICSP header. Check the PIC for bent pins. Make sure the PICKit software is correctly identifying the part. Re-try the programming process making sure to erase. A Blank Check might also be a good idea after an erase in this case to be sure things are really erasing. Try another part- if things work, then you have a bad part.

2.3.2.2. PICKit 2 Programming Example

When you purchased you programmer, you should have received either a disk with the software, or a link to download the PICKit2 software.
You can also get a copy directly from MicroChip here:
http://ww1.microchip.com/downloads/en/DeviceDoc/PICkit%202%20v2.61.00%20Setup%20A.zip

Check on this page for updated information, other tools, etc.
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805

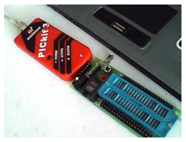
Here is a funny thing- the PICKit 2 Programming is exactly the same as the PICKit 3 (as above) just it will say PICKit 2 wherever it says PICKit 3 as we show in the screen shots or directions above. Just follow those exact same steps- Microchip intentionally duplicated the PICKit 2 experience on the PICKit 3 so this is by design.

2.3.3. Proprietary

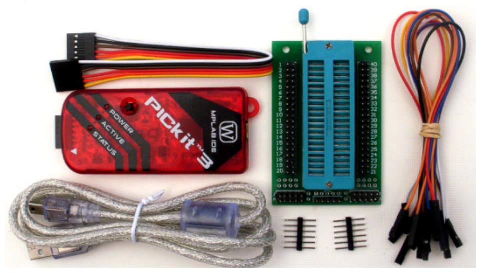
Proprietary (i.e. NOT PICkit compatible) programmers can be used also as long as the support the PIC parts we need to program. Unfortunately we cannot give you screen shots or examples. Take a look at the process for the PICKits and it should be VERY similar process:

- Install programming software,
- Attach to device to be programmed (it may be a ZIF only solution)\
  - If it is in circuit via the ICSP, you will need to check to see if the programmer powers the chip or if you need to apply external power (like the board power system).
- Recognize the right part or manually configure it (some progammers may not be able to detect the chip type and need to be told manually what part you want to program.
- ERASE
- Load or Import HEX file. Note: it is possible that some programmers do NOT accept HEX file as a file format. It may need a BIN file. There are utilities that may be included with your programmer software to convert a HEX file to binary (HEX2BIN is a very commonly available utility- just Google it).
- Program / Write
- Verify
- BLINK LIGHTS!!

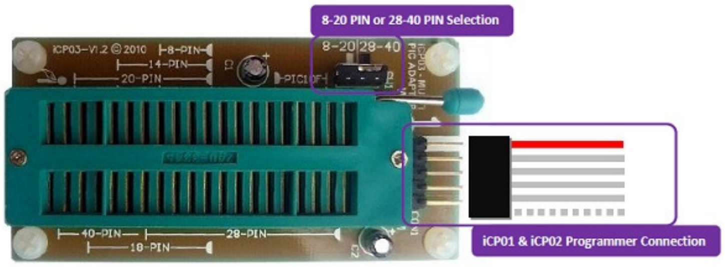
2.4. ZIF Sockets



ZIF stands for Zero Insertion Force. A ZIF socket allows easier installation and removal for applications such as programmers where you are programming loose parts. This is needed for some of the Renard board designs like the SS16 or Headblinker boards that do not support on-board programming through an ICSP connector. Some programmers come with a ZIF capability built in. However, many of the common PICKit or compatible designs do not natively support handling programming the PIC out of the board. For those cases you might want to consider getting a ZIF socket that works with your programmer. When you purchase your programmer, often there are "bundled" options like this PICKit 3 and ZIF kit:



If you already have your PICKit programmer or equivalent and want to add off-board programming capability you can find many offerings like:


8-20 PIN or 28-40 PIN Selection
ICP03-V1.2 © 2010
8-PIN
14-PIN
20-PIN
40-PIN
28-PIN
18-PIN
8-20
28-40
iCP01 & iCP02 Programmer Connection

(this ZIF is easy to configure and use with just about ANY size thru-hole PIC, and any PICKit that supports ICSP capability. This is the ZIF the author uses for situations where there is no ICSP connector available. It is easy to use and very flexible.)

2.5. MPLab

MPLab is the IDE (Integrated Development Environment) from Microchip that allows the Renard control software to be compiled into firmware that is programmed into the PIC. We assume that you have successfully downloaded and installed the MPLAB code from the Microchip website and that the software is at version v8.86 or newer. To use MPLabs to program the PIC directly, you need to have a PICKit or compatible programmer (as described before), or one of the other programmers that MPLabs supports (check with Microchip for details). MPLabs is most useful for compiling and programming from source, rather than to program a pre-compiled HEX. Using MPLabs to program a HEX file can be done but is much more cumbersome than using the PICKit Debug/Programming software. However, the setup required for HEX programming is also needed to do a compile/program so the steps are useful (at least for the configuration steps).

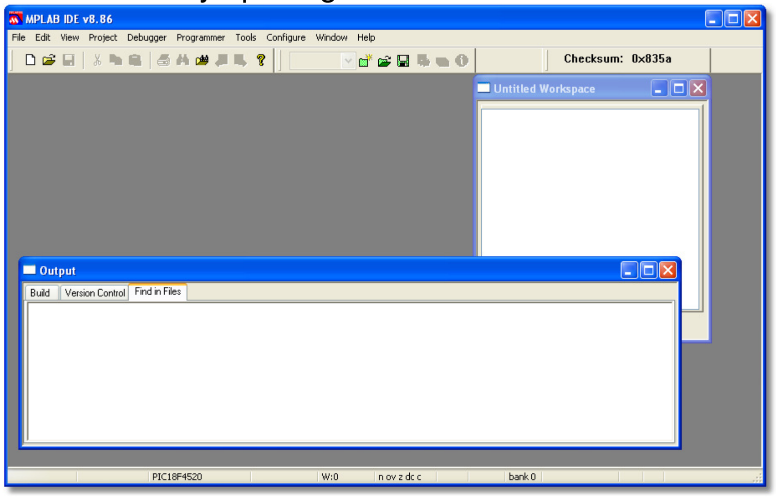
2.5.1. Installing MPLabs

Our recommendations is to use the older MPLab (8.6.xx) from the archives at Microchip archives because in our experience it is easier to use, supports more devices, and is less picky about the programming style than the newer version. Most of the Renard code has been developed using version 8 or older of MPLabs and they may not translate well to the newer MPLab X version without some code changes.

Additionally, in order to avoid path issues with files, we highly recommend installing MPLabs on the default locations on Drive C. In theory, it is possible to adapt to other locations, but MPLabs hardcodes drive letters.

2.5.2. HEX handling with MPLabs

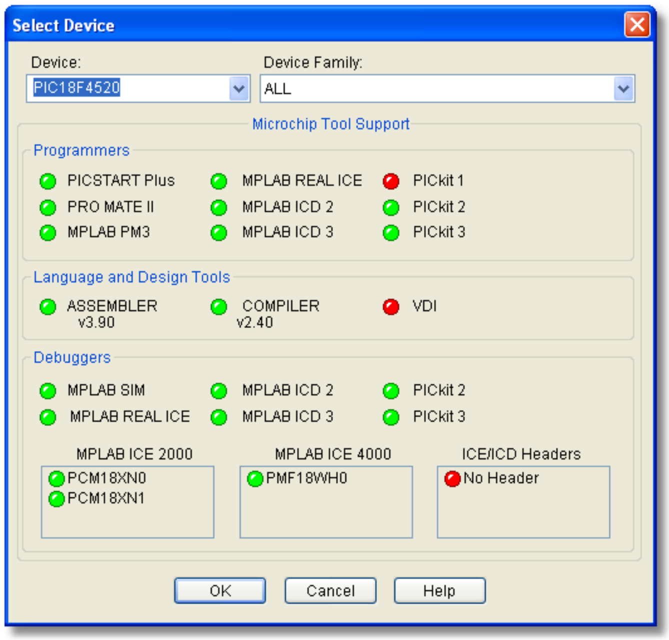
First start of by opening MPLAB



Initial MPLAB IDE screen

If you do not see the output screen it can be opened using the View Menu and the second option will be Output.

Step 1 Configure Device


Select Device

Device: PIC18F4520
Device Family: ALL

Microchip Tool Support

Programmers

🟢 PICSTART Plus	🟢 MPLAB REAL ICE	🔴 PICkit 1
🟢 PRO MATE II	🟢 MPLAB ICD 2	🟢 PICkit 2
🟢 MPLAB PM3	🟢 MPLAB ICD 3	🟢 PICkit 3

Language and Design Tools

🟢 ASSEMBLER v3.90	🟢 COMPILER v2.40	🔴 VDI

Debuggers

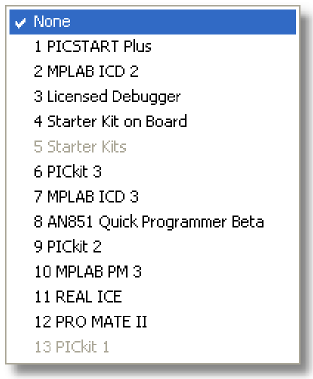
🟢 MPLAB SIM	🟢 MPLAB ICD 2	🟢 PICkit 2
🟢 MPLAB REAL ICE	🟢 MPLAB ICD 3	🟢 PICkit 3

MPLAB ICE 2000	MPLAB ICE 4000	ICE/ICD Headers
🟢 PCM18XN0	🟢 PMF18WH0	🔴 No Header
🟢 PCM18XN1		

Select Device Screen

Use the device drop-down box to select the PIC chip you will be programming. All of the other boxes and controls can be left to default.

Step 2 Configure Programmer

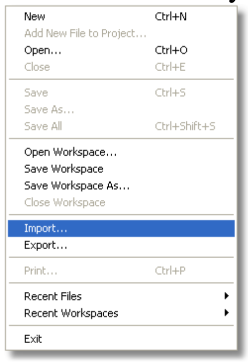


Programmer Menu

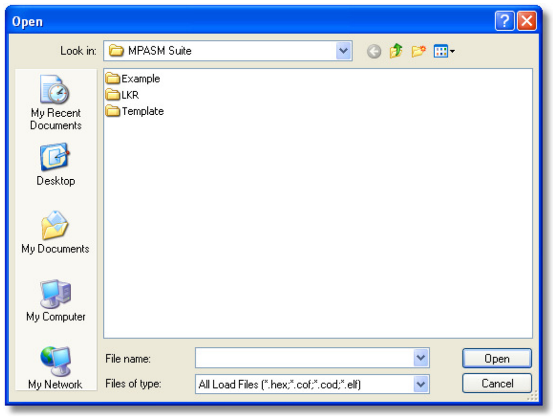
From this menu select the programmer you will be using. Once selected the little check mark will be positioned on the programmer you selected.

Step 3 Importing Pre-compiled HEX file

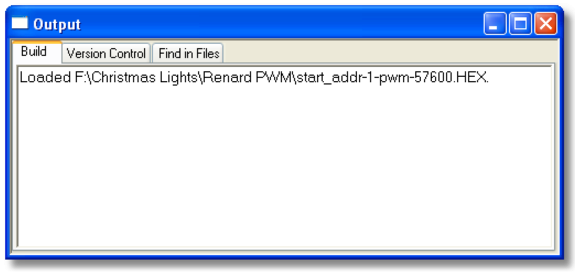
On the file menu you will see a menu item titled Import:



Select this option and the IDE will open a standard Windows Open window that will allow you to traverse to the folder on your computer where your HEX files are located:



Once you have located and selected your hex file the MPLAB software will display in the output window that the file has successfully been loaded.



```
Loaded F:\Christmas Lights\Renard PWM\start_addr-1-pwm-57600.HEX.
```

Step 4 Erase (if your part has been programmed before), Program and Verify PIC Chip

At this point, you part is programmed and ready to use. If you have an ICSP enabled board, you should now be able to power up and start using your board. For other solutions, you need to install the part in the appropriate spot of the board and proceed with any other programming and/or assembly you need to do.

2.5.3. Source handling with MPLabs

The real use for MPLabs is to be able to make modifications to the source code (like the options we discussed previously in the section “Firmware Options”) and compile then program the resulting HEX. One of the most common reasons for re-compiling a project is to change the start address or baud rate, and rarely for a bug fix. This section will walk you through the basic steps of modifying, compiling, and re-programming a PIC chip.

2.5.3.1. Rebuilding and re-compiling Renard-Plus Projects

Setting up MPLabs and creating projects is a reasonably complicated process. The best advise we can give you is to spend some time on the MicroChip website exploring the tutorials there to learn about how MPLabs works and its use in general. We will try to provide some common “recipes” for “cooking” your own firmware but a knowledge of MPLabs will be required and is beyond our capability to provide in this document. We promise to try to improve this as we go, but again, because of the complexity, the MPLabs knowledge is best obtained from the provider- MicroChip.

TBD: This section applies to RP source and is a work in progress and will cover these topics and hopefully more:

- Detail source installation and access via IDE.
- Describe ASM variables.
- Show common settings.
- Screenshots of successful steps.
- Troubleshooting for issues.

3. Notes

This area provided for you to keep notes on the programming process as you see fit.