ReadMe.Txt

**Intro**

I needed a small body of code to decode data from my uBlox Neo 7m GPS receiver which is used on my Flying Squirrel High Altitude Balloon projects. The popular TinyGPS & TinyGPS++ libraries are very useful, but were taking up too much program memory on my Arduino boards, leaving not enough memory to add some extra functions to the project. Thus this library was born. This library uses 2K less program memory space than the TinyGPS library.

The uBlox GPS receivers, like many GPSs default to a standard Ascii message format called NEMA. However the uBlox receivers also have their own binary message format (uBX protocol) which takes much less program code to decode the message.

UBX has many message types, but the NAV-PVT (Navigation Position Velocity Time Solution) message has most all the GPS data you will ever need. The uBX messages are documented in this PDF document: <https://www.u-blox.com/sites/default/files/products/documents/u-blox7-V14_ReceiverDescrProtSpec_%28GPS.G7-SW-12001%29_Public.pdf> The NAV-PVT message is documented on page 160 of this manual.

The uBlox8 GPS is similar and should also be compatible with this library. (But I have not tested this)

**What this library does.**

Decodes the uBlox uBX binary message NAV-PVT which has the GPS data you will need for most navigation purposes. This message decoder is much more efficient than decoding NEMA data at the cost of some additional GPS receiver setup.

This library is designed to work with uBlox Neo 7 Series GPS receivers and is not compatible with other brands that send NEMA data.

**Features**:

Lightweight on memory, much smaller than TinyGps(++) library. 2+K less than TinyGps.

Not dependent on Hardware Serial and will work as well with Software Serial or other Serial Interface. The serial data stream is handled in your own code, so there is no dependency in this library.

Portable code runs on Arduino 8 bit and 32 bit boards such as STM32F103 (aka Blue pill boards)

**GPS Setup**

The Neo 7 GPS has to be configured to send the uBX message NAV-PVT. It is also desirable to disable the NEMA messages sent by the uBlox GPS by default. Disabling NEMA messages will streamline the serial output from the GPS making decoding faster because there is less data sent to the micro controller board over the serial line.

There are two ways to configure the uBlox 7m GPS to do this:

1. Use the uBlox u-center application to configure the GPS to send the NAV-PVT & disable the NEMA messages and save the configuration to the GPS flash memory.

This video has a good explanation of how to set up your GPS: <https://www.youtube.com/watch?v=TwhCX0c8Xe0> If you want to skip the introductory information, jump to the 5 minute mark in the video.

Note the video shows configuring a different uBx message, but the procedure is the same.

This method will save the most memory in your program because the GPS is already setup, so no code needs to be used to set up the GPS at run time. The downside of this is that is you have to use the u-center program, which you have to learn, to configure the GPS.

2. Use the functions shown in the example program (uBloxGPSExample.ino) to configure the GPS receiver. This method is easy to use. You may need to modify the code very slightly to use your serial interface of choice (The example uses Serial2). These commands in the example will configure your GPS, but does not save the configuration, so you will have to call the setup functions each time the GPS is powered up. The downside of this method is that is uses a few hundred extra bytes of program memory. If space is not a concern, this is the easiest way to go.

These example setup functions assume the GPS is in its factory default condition, so if this is not true, you may want to use method #1 instead.

Using the class to get GPS Data.

To use this library put these lines at the top of your sketch.

#include "uBloxGPS.h"

uBloxGPS uBlox; // The uBloxGPS object.

The uBloxGPS object works by feeding the GPS data stream into the object one character at a time like this:

while (Serial2.available()) // Data from the GPS

{

uint8\_t ch = Serial2.read(); // Get next character to process

uBlox.FeedMe(ch); // Feed the serial data into the uBloxGPS object.

}

Each time a NAV-PVT message has been decoded the variable uBlox.PacketNumber will increment by one. You can poll the class by checking to see if PacketNumber has changed, and if so examine the values of the class for information.

Not all valid NAV-PVT messages are valid positions. This is especially true when the GPS is waking up from an extended period of time without power and has to re-acquire the satellites. The re-acquisition of the satellites can take some time depending on your receiver, antenna, view of the sky and other factors.

The class value uBlox.FixNumber will increment each time a valid position fix has been obtained and you can poll this value for new valid position fixes. Also you can poll with the function uBlox.IsValidFixPosition(), which will return true if the most recently decoded message was a valid fix.

The details from the NAV-PVT message are stored in the structure uBlox.PVT and can be directly read as needed. The individual fields are documented in in the uBX protocol spec. The PVT structure will be overwritten on each successfully decoded message, so you may need to copy the needed data before feeding more serial data into the FeedMe() function.

You can also register a callback function, to be called each time a new valid position has been decoded.

Your callback function should look like this example:

void PositionUpdateCallback(uBloxGPS \*gps) // Function name can be any name

{

// Do something with the object data passed into this function

}

Do enable the call back call:

uBlox.SetPositionUpdateCallbackFunction(&PositionUpdateCallback); // Use the same function name

Disable the call back by calling:

uBlox.SetPositionUpdateCallbackFunction(NULL); // Pass a null function pointer to disable.

I hope you find this code useful.