- Home
- Libraries
    - TinyGPS
    - NewSoftSerial
    - Streaming
    - PString
    - Flash
    - PWMServo
    - IridiumSBD
    - TinyGPS++
- About
- Projects
    - The Reverse Geocache™ Puzzle
        - Building a Puzzle Box
        - Buying a Puzzle Box
        - More Puzzle Box Tales

Email: Subscribe! Admin RSS Subscribe: RSS feed

Arduiniana
Arduino wisdom and gems by Mikal Hart

TinyGPS++

A *NEW* Full-featured GPS/NMEA Parser for Arduino

TinyGPS++ is a new Arduino library for parsing NMEA data streams provided by GPS modules.

Like its predecessor, TinyGPS, this library provides compact and easy-to-use methods for extracting position, date, time, altitude, speed, and course from consumer GPS devices.

However, TinyGPS++'s programmer interface is considerably simpler to use than TinyGPS, and the new library can extract arbitrary data from any of the myriad NMEA sentences out there, even proprietary ones.

Download and Installation

To install this library, download here, unzip the archive into the Arduino "libraries" folder, and restart Arduino. You should rename the folder "TinyGPSPlus".

Download

History

TinyGPS++ is the immediate inheritor of TinyGPS, a popular compact parser that is used in Arduino installations around the world. TinyGPS++ is not quite as 'tiny' as its older sibling, but its powerful and extremely easy-to-use new object model and useful new feature set make it an attractive alternative.

Usage

Let's say you have an Arduino hooked to an off-the-shelf GPS device and you want to display your altitude. You would simply create a TinyGPS++ instance like this:

```
#include "TinyGPS++.h"
TinyGPSPlus gps;
```

Repeatedly feed it characters from your GPS device:

```
while (ss.available() > 0)
  gps.encode(ss.read());
```

Then query it for the desired information:

```
if (gps.altitude.isUpdated())
  Serial.println(gps.altitude.meters());
```

Differences from TinyGPS

Although TinyGPS++ shares much the same compact parsing engine with TinyGPS, its programmer interface is somewhat more intuitive. As a simple example, here's how easy it is to print out the current latitude, longitude, and altitude in TinyGPS++:

```
Serial.print("LAT=");  Serial.println(gps.location.lat(), 6);
Serial.print("LONG="); Serial.println(gps.location.lng(), 6);
Serial.print("ALT=");  Serial.println(gps.altitude.meters());
```

Both libraries extract basic position, altitude, course, time, and date, etc. from two common NMEA sentences, **$GPGGA** and **$GPRMC**. But there are a number of other interesting sentences out there, both NMEA-defined and vendor-proprietary, just waiting to be harvested.

Consider the obscure **$GPRMB**, for example, which provides "recommended minimum navigation information" if you have a destination waypoint defined.

```
$GPRMB,A,4.08,L,EGLL,EGLM,5130.02,N,00046.34,W,004.6,213.9,122.9,A*3D
```

With TinyGPS++ it is now possible to extract just the "L" in the third field (it means "steer **L**eft!"). It's easy with the new TinyGPSCustom watcher object:

```
TinyGPSCustom steerDirection(gps, "GPRMB", 3);
...
Serial.print(steerDirection.value()); // prints "L" or "R"
```

Naturally, this extra functionality comes at some cost. TinyGPS++ consumes somewhat more memory than TinyGPS, and it's interface is incompatible. So how to decide whether to update? Here's a guide:

Consider TinyGPS++ over TinyGPS if:

- Compatibility with existing code (using TinyGPS) isn't necessary.
- Your sketch is not close to reaching RAM or flash resource limits.
- You are running on Due or processor which can take advantage of the higher precision of 64-bit "double" floating-point.
- You prefer the more intuitive object model.
- You need to query for NMEA data beyond the basic location, date, time, altitude, course, speed, satellites or hdop.

Feeding the Hungry Object

To get TinyGPS++ to work, you have to repeatedly funnel the characters to it from the GPS module using the **encode()** method. For example, if your GPS module is attached to pins 4(RX) and 3(TX), you might write code like this:

```
SoftwareSerial ss(4, 3);
void loop()
{
  while (ss.available() > 0)
    gps.encode(ss.read);
  ...
```

After the object has been "fed" you can query it to see if any data fields have been updated:

```
  if (gps.location.isUpdated())
  {
    Serial.print("LAT="); Serial.print(gps.location.lat(), 6);
    Serial.print("LNG="); Serial.println(gps.location.lng(), 6);
  }
} // end loop()
```

The TinyGPS++ Object Model

The main TinyGPS++ object contains several core sub-objects:

- **location** – the latest position fix
- **date** – the latest date fix (UT)
- **time** – the latest time fix (UT)
- **speed** – current ground speed
- **course** – current ground course
- **altitude** – latest altitude fix
- **satellites** – the number of visible, participating satellites
- **hdop** – horizontal diminution of precision

Each provides methods to examine its current value, sometimes in multiple formats and units. Here's a complete list:

```
Serial.println(gps.location.lat(), 6); // Latitude in degrees (double)
Serial.println(gps.location.lng(), 6); // Longitude in degrees (double)
Serial.print(gps.location.rawLat().negative ? "-" : "+");
Serial.println(gps.location.rawLat().deg); // Raw latitude in whole degrees
Serial.println(gps.location.rawLat().billionths);// ... and billionths (u16/u32)
Serial.print(gps.location.rawLng().negative ? "-" : "+");
Serial.println(gps.location.rawLng().deg); // Raw longitude in whole degrees
Serial.println(gps.location.rawLng().billionths);// ... and billionths (u16/u32)
Serial.println(gps.date.value()); // Raw date in DDMMYY format (u32)
Serial.println(gps.date.year()); // Year (2000+) (u16)
Serial.println(gps.date.month()); // Month (1-12) (u8)
Serial.println(gps.date.day()); // Day (1-31) (u8)
Serial.println(gps.time.value()); // Raw time in HHMMSSCC format (u32)
Serial.println(gps.time.hour()); // Hour (0-23) (u8)
Serial.println(gps.time.minute()); // Minute (0-59) (u8)
Serial.println(gps.time.second()); // Second (0-59) (u8)
Serial.println(gps.time.centisecond()); // 100ths of a second (0-99) (u8)
Serial.println(gps.speed.value()); // Raw speed in 100ths of a knot (i32)
Serial.println(gps.speed.knots()); // Speed in knots (double)
Serial.println(gps.speed.mph()); // Speed in miles per hour (double)
Serial.println(gps.speed.mps()); // Speed in meters per second (double)
Serial.println(gps.speed.kmph()); // Speed in kilometers per hour (double)
Serial.println(gps.course.value()); // Raw course in 100ths of a degree (i32)
Serial.println(gps.course.deg()); // Course in degrees (double)
Serial.println(gps.altitude.value()); // Raw altitude in centimeters (i32)
Serial.println(gps.altitude.meters()); // Altitude in meters (double)
Serial.println(gps.altitude.miles()); // Altitude in miles (double)
Serial.println(gps.altitude.kilometers()); // Altitude in kilometers (double)
Serial.println(gps.altitude.feet()); // Altitude in feet (double)
Serial.println(gps.satellites.value()); // Number of satellites in use (u32)
Serial.println(gps.hdop.value()); // Horizontal Dim. of Precision (100ths-i32)
```

Validity, Update status, and Age

You can examine an object’s value at any time, but unless TinyGPS++ has recently been fed from the GPS, it should not be considered valid and up-to-date. The **isValid()** method will tell you whether the object contains any valid data and is safe to query.

Similarly, **isUpdated()** indicates whether the object’s value has been updated (not necessarily *changed*) since the last time you queried it.

Lastly, if you want to know how stale an object’s data is, call its **age()** method, which returns the number of milliseconds since its last update. If this returns a value greater than 1500 or so, it may be a sign of a problem like a lost fix.

Debugging

When a TinyGPS++ sketch fails, it’s usually because the object received an incomplete NMEA stream, or perhaps none at all.

Fortunately, it’s pretty easy to determine what’s going wrong using some built-in diagnostic methods:

- **charsProcessed()** – the total number of characters received by the object
- **sentencesWithFix()** – the number of $GPRMC or $GPGGA sentences that had a fix
- **failedChecksum()** – the number of sentences of all types that failed the checksum test
- **passedChecksum()** – the number of sentences of all types that passed the checksum test

If your sketch has been running a while but charsProcessed() is returning 0, you likely have a problem with your wiring or serial connection. (If data never arrives from the GPS unit, it stands to reason it’s not getting to TinyGPS++.) I often insert a little debug clause into my GPS sketches detects this condition then prints out the incoming stream:

```
// Debug: if we haven't seen lots of data in 5 seconds, something's wrong.
if (millis() > 5000 && gps.charsProcessed() < 10) // uh oh
{
  Serial.println("ERROR: not getting any GPS data!");
  // dump the stream to Serial
  Serial.println("GPS stream dump:");
  while (true) // infinite loop
    if (ss.available() > 0) // any data coming in?
      Serial.write(ss.read());
}
```

Another common failure is when the sentences sent to TinyGPS++ are incomplete. This usually happens when you retrieve the characters from the GPS so slowly or infrequently that some are lost. The symptom is easy to spot: **checksum failure**.

Explanation: Every NMEA sentence ends with a numeric field that represents a mathematical summing of all the characters in the sentence. It’s there to ensure data integrity. If this number doesn’t match the actual sum (perhaps because some characters went awry), TinyGPS++ simply discards the entire sentence and increments an internal “checksum failed” counter. You can read this counter with:

```
Serial.print("Sentences that failed checksum=");
Serial.println(gps.failedChecksum());

// Testing overflow in SoftwareSerial is sometimes useful too.
Serial.print("Soft Serial device overflowed? ");
Serial.println(ss.overflow() ? "YES!" : "No");
```

If the checksum counter is continually incrementing, you have a problem. (Hint: don’t use **delay()** in your sketch.)

Custom NMEA Sentence Extraction

One of the great new features of TinyGPS++ is the ability to extract arbitrary data from *any* NMEA or NMEA-like sentence. Read up on some of the interesting sentences there are out there, then check to make sure that your GPS receiver can generate them.

The idea behind custom extraction is that you tell TinyGPS++ the sentence name and the field number you are interested in, like this:

```
TinyGPSCustom magneticVariation(gps, "GPRMC", 10)
```

This instructs TinyGPS++ to keep an eye out for $GPRMC sentences, and extract the 10th comma-separated field each time one flows by. At this point, magneticVariation is a new object just like the built-in ones. You can query it just like the others:

```
if (magneticVariation.isUpdated())
{
  Serial.print("Magnetic variation is ");
  Serial.println(magneticVariation.value());
}
```

Establishing a fix

TinyGPS++ objects depend on their host sketch to feed them valid and current NMEA GPS data. To ensure their world-view is continually up-to-date, three things must happen:

1. You must continually feed the object serial NMEA data with **encode()**.
2. The NMEA sentences must pass the checksum test.
3. For built-in (non-custom) objects, the NMEA sentences must self-report themselves as valid. That is, if the $GPRMC sentence reports a validity of “V” (void) instead of “A” (active), or if the $GPGGA sentence reports fix type “0″ (no fix), then the position and altitude information is discarded (though time and date are retained).

It may take several minutes for a device to establish a fix, especially it has traveled some distance or a long time has elapsed since its last use.

Distance and Course

If your application has some notion of a “waypoint” or destination, it is sometimes useful to be able to calculate the distance to that waypoint and the direction, or “course”, you must travel to get there. TinyGPS++ provides two methods to get this information, and a third (**cardinal()**) to display the course in friendly, human-readable compass directions.

```
const double EIFFEL_TOWER_LAT = 48.85826;
const double EIFFEL_TOWER_LNG = 2.294516;
double distanceKm =
  TinyGPSPlus.distanceBetween(
    gps.location.lat(),
    gps.location.lng(),
    EIFFEL_TOWER_LAT,
    EIFFEL_TOWER_LNG) / 1000.0;
double courseTo =
  TinyGPSPlus.courseTo(
    gps.location.lat(),
    gps.location.lng(),
    EIFFEL_TOWER_LAT,
    EIFFEL_TOWER_LNG);
Serial.print("Distance (km) to Eiffel Tower: ");
Serial.println(distanceKm);
Serial.print("Course to Eiffel Tower: ");
Serial.println(courseTo);
Serial.print("Human directions: ");
Serial.println(TinyGPSPlus.cardinal(courseTo));
```

Library Version

You can retrieve the version of the TinyGPS++ library by calling the static member **libraryVersion()**.

```
Serial.println(TinyGPSPlus::libraryVersion());
```

Sample Sketches

TinyGPS++ ships with several sample sketch which range from the simple to the more elaborate. Start with **BasicExample**, which demonstrates library basics without even requiring a GPS device, then move onto **FullExample** and **KitchenSink**. Later, see if you can understand how to do custom extractions with some of the other examples.

Acknowledgements

Thanks go out to the many Arduino forum users for outstanding help in testing and popularizing TinyGPS, this library’s predecessor. Thanks especially to Maarten Lamers, who wrote the wiring library that originally gave me the idea of how to organize TinyGPS++.

All input is appreciated.

Mikal Hart