SirfGPSTweaker documentation

www.gosiewski.pl/marcin/programming

Contents

What is SirfGPSTweaker

Application for displaying output from NMEA or SIRFBINARY based GPS device. It has autodetection of data format (so it works both in NMEA or SIRF with on the fly detection). It can display statistics and data panel for cockpit use. Cockpit mode includes history graphs for speed and altitude. Programm is able to tweak GPS receivers by changing the operation mode or resetting the device. It can save output from GPS to a dump file and save comprehensive debug information. .

Downloads

Application: SirfGPSTweaker.zip
PDF Documentation: SirfGPSTweaker.pdf
HTML Documentation: SirfGPSTweaker.mht

Concepts

I wanted to create the application which can be used for two needs: to debug, fine tune GPS receiver as well as examine operation characteristics of GPS receivers in various conditions, and as a dashboard computer for a journey with logging functionality & speed / altitude graphs. I was trying to use some available software like VisualGPS CE or SirfDemo. None of them was good for me – so I decided to write my own.

The basic assumptions are:

- Application decodes both NMEA and SIRF BINARY code. It can find which protocol to use on the fly, you do not have to set any modes in advance. It is easy to setup the device.

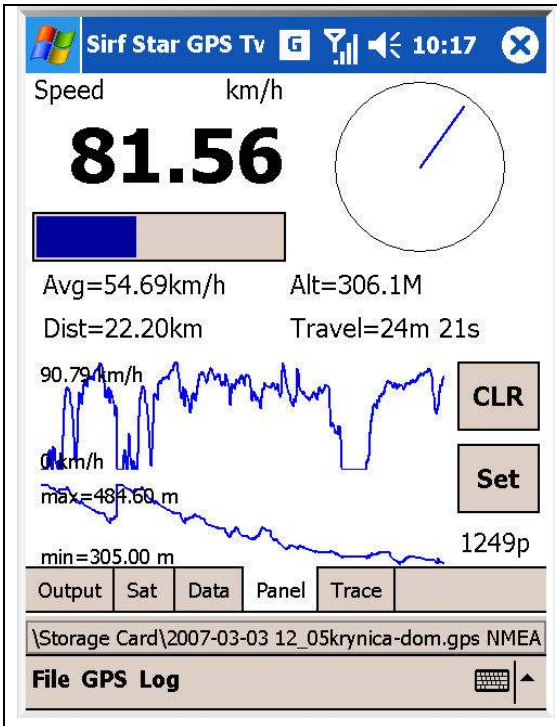
Any device capable of NMEA output can be used with the program. It does not need to be SIRF based however you can do more fine tuning with SIRF.

- It can give a lot of debugging information on user level: good screen with BOTH signal level and satellites azimuth/elevation on one screen, screen with online dump of GPS output and lots of statistics (bytes read, used, discarded, sentences processed successfully etc)
- It has a good on-board computer dashboard for a car, showing all trip information on one screen: current speed (number&graph), heading, time of travel, distance traveled etc. as well as nice graphs with speed /altitude history.
- Ability to record all data coming from GPS receiver to a file or use file as input
- Records detailed debug log showing all connection details, incoming bytes, information on decoding them as sentences or discarding as invalid etc. – very detailed information.
- I do not intend this to be a mapping application. I am not planning to add mapping functionality
- Application remembers it's connection settings and reconnects to a last recently used GPS data source (com port or file) upon startup.

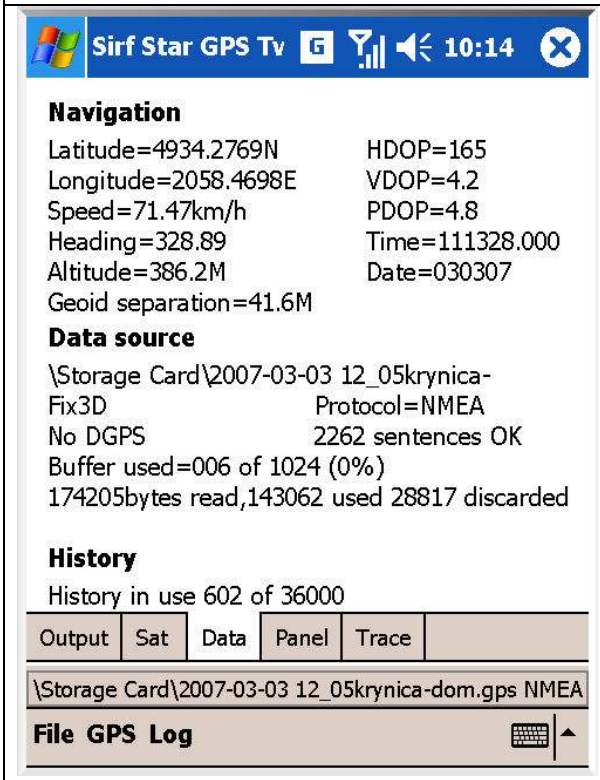
Requirements

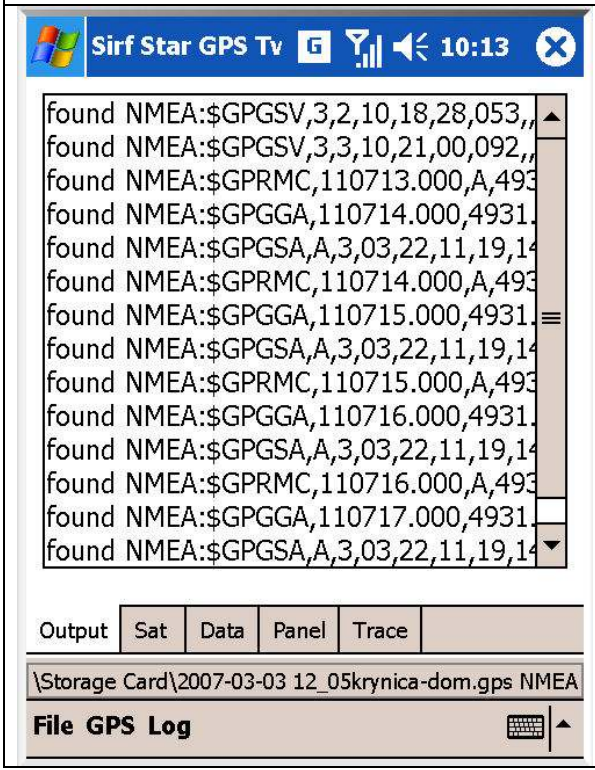
- Application requires .NET framework 2.0 or higher. Pocket PC version can be downloaded from http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=0C1B0A88-59E2-4EBA-A70E-4CD851C5FCC4
- The same executable works on PC/Windows and Pocket PC etc.
- No need to install the application, just copy to any folder and run it.

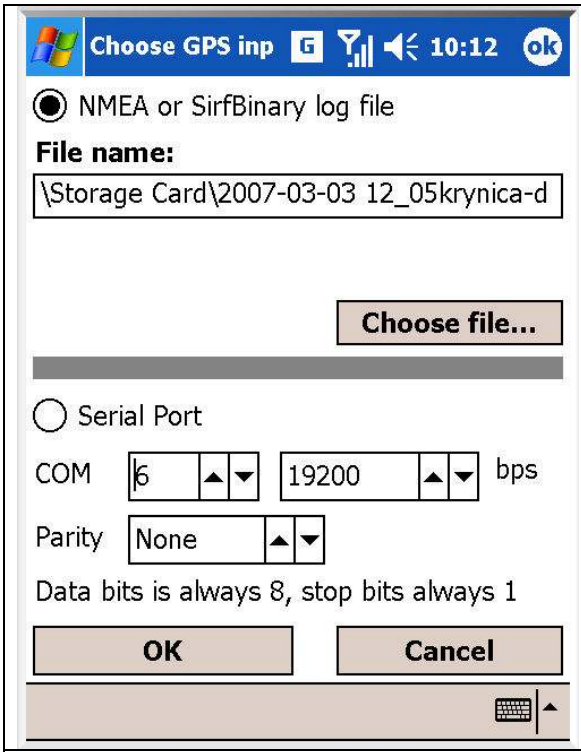
Screenshots with explaination of displayed data



Speed = current speed obtained from GPS. Below is the graph showing current speed. It is scaled from 0 to 200km/h.
On the right – **compass**. North relative to current travel direction is shown by the blue pointer.
Avg = average speed from beginning of journey or history cleared (in km/h). I can be reset by CLR button. Time when GPS is not transmitting (File->Disconnect) or pocket PC is turned off for >30 seconds is not calculated into average. However if device is still in one place (speed=0km/h) and GPS is transmitting is taken into account lowering the average.
Dist = distance traveled while online in kilometres. Distance is calculated from speed multiplied by time, not from latitude / longitude readings.
Alt = current altitude above sea level in meters.
Travel = travel time in minutes / seconds
1249p = number of history points used for

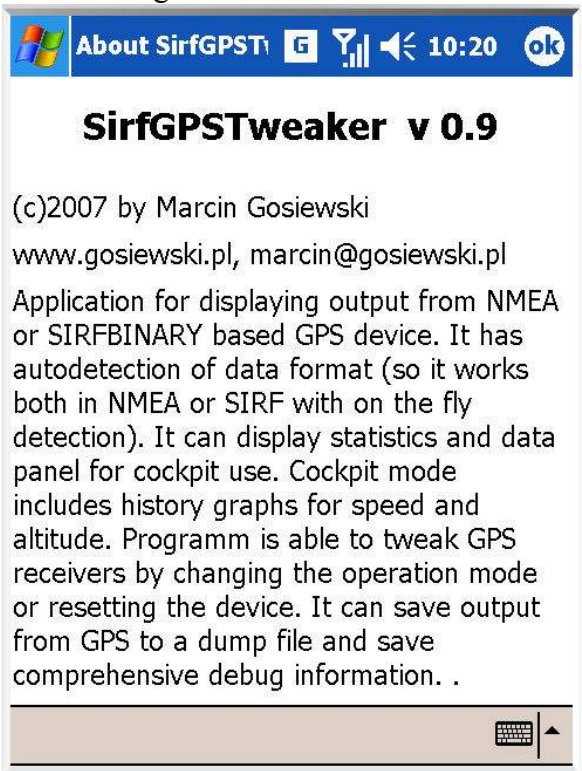
	graphs on the left side. Next point is added when GPS sends new update to application (usually every second) **Speed graph** = shows graph of speed from beginning of journey till now. If all history buffer is used (default is 10 hours of recording) the graph moves to the left discarding oldest data (so the graph will show speed during last 10 hours). Discarding oldest graph data does not affect average speed, min and max speed and other statistical information calculated.
	Navigation data consists of: **Latitude**, **Longitude**, **Speed**, **Heading** (direction of traveling) and current **Altitude** above sea level. **Geoid separation** is the correction applied to Altitude to make altitude real. If number is nonzero, than Altitude is probably correct above mean sea level, if zero it means Altitude is not corrected and can be invalid because is used from a simplified WGS-81 model. Additionally it shows: **HDOP** = horizontal dilution of precision as reported from GPS. The same with **VDOP** (vertical) and **PDOP** (3D fix dilution of precision. DOP parameters are taking into account only current satellites configuration (positions, number of sats visible) and not real time environment. **Time** & **Date** = date and time of last succesfull fix. Not current date and time. Data source is the COM port and baud rate or filename used as GPS input. **Fix3D**/Fix2D – type of last fix. Usually if >3 satellites can be used the fix is 3D it means altitude is also calculated. **DGPS** = shows Differential GPS status. No DGPS = not in use, DGPS fix time and satelliute ID is shown when in use. DGPS is additional source of information on current GPS error obtained from extra satellite (WAAS/EGNOS) or terrestal station **Protocol**= the protocol used to successfully decode last sentence. Can be SIRF binary or NMEA **Buffer used** = number of bytes left in input buffer after last fix. Nonzero number means

	that part of next sentence was read which will be probably completed and read in next update. It is normal. Large numbers however (above 100) mean there is some junk in input and both SIRF nor NMEA decoders can't find anything for themselves. **Bytes read,used,discarded** = how many bytes were read from GPS receiver, how many were used in successfully decoded SIRF or NMEA sentences an how many had to be discarded because they couldn't be recognized as valid sentences. **History in use** – how many updates are recorded in history buffer for speed & altitude & position graphs, and total size of that buffer. This does not affect logging to file. History buffer is used only for graphs. If full buffer is used, the oldest updates are discarded.
	On this screen bytes received from GPS which are forming valid sentences are shown. For NMEA – it is text output, for SIRF it is HEX dump. This is preliminary and rough debug information. Full information can be obtained from Debug Log file if recorded. This can be used to see if GPS is transmitting at all and if the transmission is anyhow useful for decoding.

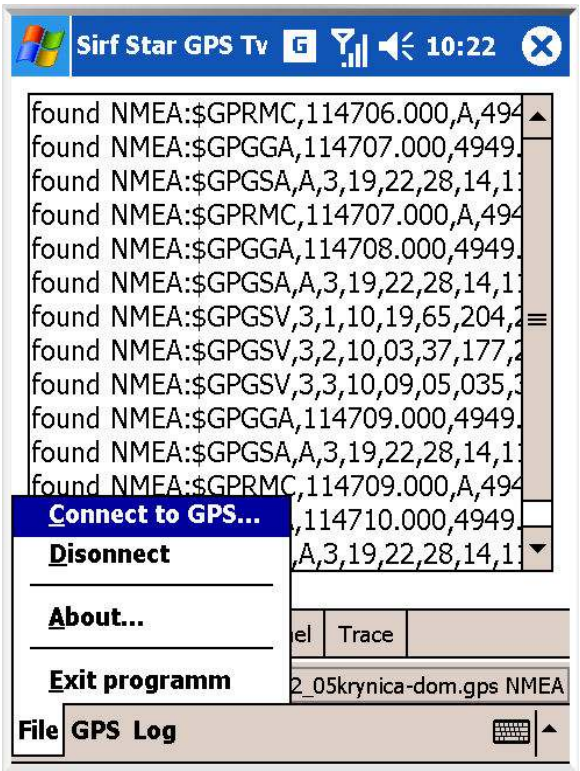


As input you can use COM port (serial) or previously recorded dump file with GPS receiver output. File can be recorded with this application or can be from any other application like VisualGPS or SIRFDemo. This screen is used to specify the source and source parameters.

About dialog

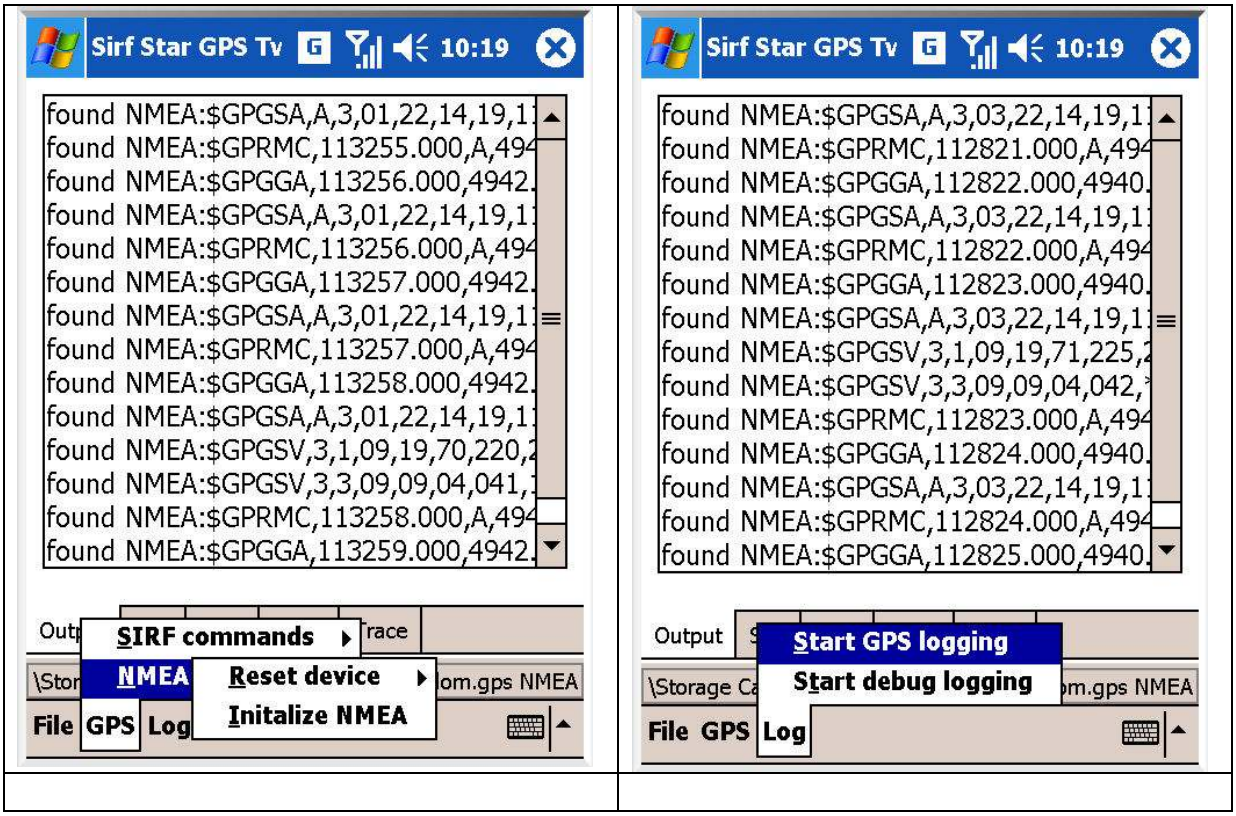


File menu



GPS commands menu

Log files menu



Notes on operation

When launched the application first searches current user's 'My Documents' folder for a file 'SirfGPSTweaker.ini' containing the previously used configuration (previously open GPS device and previously open panel etc). If found – loads the data. If not – uses default values for startup.

Known bugs

- SIRF decoder is not implemented yet.
- From NMEA only the following sentences are fully implemented: GPRMC, GPGGA, GPGSA, GPGSV, GPGLL
- GPRMB, PGRME, PGRMZ, GPBOD, GPRTE are partially implemented
- Application hangs when the device is switched on during serial transmission and the device is switched on some time later.

Technical details – source code analysis

Programm consists of a set of forms and some libraries:

Form1.cs – main screen with all tabs (output, panel, GPS data etc) & main menu
FormAbout.cs – about dialog
FormConnectionSettings.cs – dialog for choosing GPS data source

Libraries:
GPSData.cs – structure containing data obtained from last fix of GPS and tables containing historical data for Latitude, Longitude, Altitude + calculated statistics. Also contains few methods for manipulating historical data (HistoryClear, HistoryUpdate etc).

GPSDecoder.cs – class containing main decoding loop 'GetNextUpdateIfPossible()' called periodically from application. This function reads all available data from GPS receiver to the buffer, and checks the buffer for

GPSDataSource.cs – class for reading GPS data from serial port or file, opening and using gps dump log file, and debug log file.

NMEA0183.cs – class containing static functions for decoding NMEA sentences. Main function of this class is UpdateGPSData(nmea_sentence, gps_data_to_update) which takes nmea sentence string as input, and updates object of a class GPSData with decoded inforation.

SIRFBINARY.cs – identical class with static functions for decoding SIRF. It has similar UpdateGPSData(*) function

Simplified listings

Listing 1: Form1.cs

```
/*************************************************************
 * GPS Application and Libraries
 * (c) 2007 by Marcin Gosiewski
 * www.gosiewski.pl
 * marcin@gosiewski.pl
 * Do not remove the copyright note!
 *************************************************************/
namespace SirfGPSTweaker
{
    public partial class Form1 : Form
    {
        GPSDataSource dataSource;
        GPSDecoder gpsDecoder = new GPSDecoder();

        private void timer1_Tick(object sender, EventArgs e)
        {
                gpsDecoder.GetNextUpdateIfPossible(dataSource))

                switch (this.tabControl1.SelectedIndex)
                {
                    case 0: // Panel OUTPUT
                        { //tu rysujemy zawartość ekranu
                          //na bazie obiektu gpsDecoder.currentGPSData
                    case 2: // Panel DATA
                        {
                    case 3: // Panel PANEL
                        {
                    case 4:// Panel TRACE
                        break;
                    default:
                        break;
                }
            }
        }
```

```
    }
} // namespace
```

Listing 2: GPSDecoder.cs

```
GPS Decoding functionality, Class where the GPS decoding functionality
lies. It maintains buffer to read data from, holds the data read from
GPSDataSource and parses it through appriopriate NMEA or SIRFBINARY decoder
Most important method is GetNextUpdateIfPossible which reads data and
parses it.
public class GPSDecoder
{
    // The data from GPS receiver is stored here before decoding.
    public int[] buffer = new int[MAXBUFFER];
    // Holds the data currently decoded by the class.
    public GPSData CurrentGPSData = new GPSData();

    public void BufferDiscardFirstItems(int count)
    { … }

    // <summary> try to find valid NMEA sentence in buffer, decode it and
discard from buffer.
    public bool DecodeNMEA(GPSDataSource datasource)
    {
        // parse buffer to find valid NMEA sentence.
        // Than process it using static methods from NMEA0183.cs
        //and discard from buffer everything between beginnig of the buffer
        // and the sentence, including the processed sentence.

                if (NMEA0183.ChecksumValid(NMEASentence))
                {
                    //update statistics
                    NMEA0183.UpdateGPSData(NMEASentence, CurrentGPSData);
                    BufferDiscardFirstItems(j + 2);
                }
            }
        }
    }

    public bool DecodeSIRF(GPSDataSource datasource)
    {
        //  Same as DecodeNMEA but for SIRF
        if (SIRFBINARY.ChecksumValid(SIRFSentence))
        {
            // update statistics
            // update CurrentGPSData
            SIRFBINARY.UpdateGPSData(SIRFSentence, CurrentGPSData);

            BufferDiscardFirstItems(i + 8 + payloadlength);
                }
            }
        }
    }

    // Get next bytes from GPS receiver, process it via NMEA or
    // SIRFBINARY & discard when done.
    public bool GetNextUpdateIfPossible(GPSDataSource datasource)
    {
        // if we encounter full buffer at start -
```

```
        // let's discart at least part of it.
        if (buffertail == MAXBUFFER - 1)
        {
            BufferDiscardFirstItems(bytestodiscard);
        }

        // now read till input empty or buffer full
        while ((datasource.BytesToRead() > 0) && (buffer not full))
        {
            buffer[buffertail++] = datasource.ReadByte();
        }
        // now try to process all sentences in buffer
        while (DecodeSIRF(datasource)) { done = true; } // first SIRF
                                        because it is more strict in format
        while (DecodeNMEA(datasource)) { done = true; } // now NMEA, more
                                                          relaxed
    }
}
```

Listing 3 GPSData.cs

```
This is a structure holding all current and historical data obtained from
GPS. Data stored here falls into categories:
1) data from last fix (latitude, longitude, speed etc., visible satelites
etc.
2) data on the fix itself (protocol used NMEA or SIRF, etc)
3) historical data and statistics
public class GPSData
{
    // ****************************
    // current data obtained from GPS.
    // ****************************

    /// Satellites are numbered from 1, not from 0.
    public const int MAXSATELITES = 40;
    /// From which protocol the data was obtained.
    public enum Protocols {
        unknown,
        Sirf,
        NMEA
    };
    public Protocols Protocol = Protocols.unknown;
    public bool DataValid = false;
    public class SateliteInfo
    {
        public bool DataValid = false;
        /// true if satelite is in view
        public bool InView = false;
        /// true if satelite was used for last fix
        public bool InUse = false;
        /// ID of this satelite
        public int SateliteNumber = 0;
        public int Azimuth = 0;
        public int Elevation = 0;
        /// Signal To Noise ratio for this satelite.
        public int SNR = 0;
    }

    // our approach is to keep data on all satellites, not only visible
    public SateliteInfo[] Satelites = new SateliteInfo[MAXSATELITES + 1];
```

```
 public int SatelitesInView = 0;

 // fix data
 public enum FixModes {
     Fix2D,
     Fix3D,
     FixInvalid
 };
 public FixModes FixMode = FixModes.FixInvalid;
 /// Operation modes of GPS receiver.
 /// As defined in NMEA
 public enum OperationModes {
     Unknown,
     Automatic,
     ///ManualForced2D3D = force receiver to use 2D mode
     ManualForced2D3D
 };
 public OperationModes OperationMode = OperationModes.Unknown;
 ///  Number of satelites used for last fix. obtained from GPS receiver.
 public int SatelitesInUse = 0;
 ///  Dilution of precision
 public double PDOP = 0;
 public double HDOP = 0; // HDOP in meters (dilution of precision)
 public double VDOP = 0; // VDOP in meters (dilution of precision)

 // navigation data
 ///  String format of Date of last fix. Directly as received from GPS
 public string DateDMY = "";
 ///  String format of UTC time of last fix. Not corrected by timezone.
 public string TimeUTC = "";
 ///  Time of last fix (we name it current here).
 public DateTime TimeCurrentFix = new DateTime();
 ///  Time of PREVIOUS fix. Used to calculate timespan between fixes.
 public DateTime TimeLastFix = new DateTime();
 ///  Navigation status obtained from GPS. 'V' - means receiver warinig,
public string NavigationStatus = "";

 ///  Values received from GPS
 public double Latitude = 0;
 public char LatitudeNS = 'N';
 public double Longitude = 0;
 public char LongitudeEW = 'E';
 public double SpeedOverGround = 0;
 public double Heading = 0;
 public double MagneticVariation = 0;
 public char MagneticVariationEW = 'E';
 public double Altitude = 0;
 public char AltitudeUnits = 'M';
 /// Correction applied by GPS to Altitude to convert from WGS81 to
     true sea level
 /// If '0' than probably Altitude is not corrected and can show weird
     results on the sea beach.
 public double HeightOfGeoid = 0;
 public char HeightOfGeoidUnits = 'M';
 ///  If DGPS (Differential GPS) is in use.
 public bool DGPSInUse = false;
 public int DGPSCorrectionAge = 0;
 public string DGPSStationID = "";

 //*******************************
 // history and statistics
```

```
    // ********************************

    public int MAXHISTORY = 36000;
    ///  Number of valid records stored in HistorySpeed, HistoryAltitude
         etc.
    public int HistoryUsed = 0;
    ///  Helper variable for records stored in <code>HistorySpeed</code>,
<code>HistoryAltitude</code> etc.
    /// All the History* arrays are rotating, i.e. when the array is full -
next elements are stored from the beginning
    /// To calculate the index of element x (where x is negative, because 0
= current element, -1 = beforelast, etc.)
    /// We have to perform 'MODULO' calculations taking into account also
the <code>HistoryUsed</code>
    /// Helper function CalculateHistoryIndex(x) has been provided although
it is not really used in the
    /// programm.
    ///
    /// index = (x + HistoryTail - HistoryUsed + MAXHISTORY) % MAXHISTORY;
    public int HistoryTail = 0;
    ///  Total number of elements added to history tables. If we are over
the size of MAXHISTORY it means some of thenm were already discarded
    /// This can be used to determine if we have consumed all history
elements to draw on screen. We cannot use the HistoryUsed because
    /// when it reaches MAXHISTORY it does not increment anymore
    public int HistoryTotalProcessed = 0;

    /// Tables holding historical data.Next element added every update.
    /// All the History* arrays are rotating, i.e. when the array is full -
        next elements are stored from the beginning
    /// To calculate index:
    ///
    /// index = (x + HistoryTail - HistoryUsed + MAXHISTORY) % MAXHISTORY;
    ///
    public double[] HistorySpeed = null;
    public double[] HistoryAltitude = null;
    public double[] HistoryLatitude = null;
    public double[] HistoryLongitude = null;
    public double HistorySpeedMax = 0;
    public double HistoryAltitudeMin = 99999;
    public double HistoryAltitudeMax = 0;
    public double HistoryLatitudeMin = 360;
    public double HistoryLatitudeMax = 0;
    public double HistoryLongitudeMin = 360;
    public double HistoryLongitudeMax = 0;
    public double HistoryAverageSpeed = 0;
    public double HistoryTimeCoveredSeconds = 0;
    public long HistoryGPSSentencesProcessed = 0;
    public long HistoryGPSBytesRead = 0;
    /// Number of bytes from GPS used for decoding (identified as valid)
    public long HistoryGPSBytesUsed = 0;
    /// Number of bytes read from GPS discarded (invalid sentences, bad
    public long HistoryGPSBytesDiscarded = 0;
    ///  Total distance traveled since history is recorded.
    /// Note:
    /// 1) This is calculated from Speed and TimeBetweenFixes, not
Latitude/Longitude.
    /// 2) Distances traveled with poor reception and GPS switched off for
more than 20 seconds are excluded
    /// that the maximum value is not any more in the table at some moment.
    public double HistoryDistance = 0;
```

```
    public void HistoryClear(int newsize)
    { }
    public void HistoryClear()
    { }

    /// Adds new history record if new data is available from GPS
    /// This function adds new record to all history arrays.
    ///
    /// 1) first checks time span between TimeLastUpdate and
       <code>TimePreviousUpdate</code>.
    ///    if the timespan is too long (i.e. GPS was switched off or poor
        receiption) the History is not updated
    ///
    /// 2) Arrays that are updated by this function:
    /// <code>HistorySpeed</code>, <code>HistoryAltitude</code>,
        <code>HistoryLongitude</code>, <code>HistoryAltitude</code>
    /// All the arrays are rotating FIFO buffers.
    ///
    /// 3) To use the arrays you have to correctly calculate index of the
        element. See discussion in <code>CalculateHistoryIndex method.
    public void HistoryUpdate()
    {
        TimeSpan TimeSinceLastUpdate = TimeCurrentFix - TimeLastFix;

        if (HistoryUsed < MAXHISTORY)
            HistoryUsed++;

        // update history FIFO arrays
        HistorySpeed[HistoryTail] = SpeedOverGround;
        if (SpeedOverGround > HistorySpeedMax) HistorySpeedMax =
SpeedOverGround;
        HistoryAltitude[HistoryTail] = Altitude;
        if (Altitude > HistoryAltitudeMax) HistoryAltitudeMax = Altitude;
        if ((Altitude < HistoryAltitudeMin) && (Altitude != 0))
HistoryAltitudeMin = Altitude; // we ignore Alt=0 which results from 2DFix
        HistoryLatitude[HistoryTail] = Latitude;
        if (Latitude > HistoryLatitudeMax) HistoryLatitudeMax = Latitude;
        if (Latitude < HistoryLatitudeMin) HistoryLatitudeMin = Latitude;
        HistoryLongitude[HistoryTail] = Longitude;
        if (Longitude > HistoryLongitudeMax) HistoryLongitudeMax =
Longitude;
        if (Longitude < HistoryLongitudeMin) HistoryLongitudeMin =
Longitude;

        // calculate distance and average speed
        HistoryTimeCoveredSeconds += TimeSinceLastUpdate.TotalSeconds; //
seconds
        HistoryDistance += SpeedOverGround *
TimeSinceLastUpdate.TotalHours; //km
        HistoryAverageSpeed = HistoryDistance / HistoryTimeCoveredSeconds *
3600; //km/h

        // move the tail of the FIFO rotating buffers
        HistoryTail = (HistoryTail + 1) % MAXHISTORY;
        HistoryTotalProcessed++;
    }
}
```

Listing 4 NMEA0183.cs, SIRFBINARY.cs is fairly identical

```
class NMEA0183
{
    public static string CalculateChecksum(string sentence)
    { }

    /// Build full NMEA sentence from a given command and parameters.
    /// Sentence includes leading '$' and valid checksum.
    public static string BuildNMEASentence(string command, string
parameters)
    { }
    public static string BuildNMEASentence(string command, string[]
parameters)
    { }

    /// Returns true if NMEA sentence has a valid checksum.
    public static bool ChecksumValid(string sentence)
    { }

    /// Main NMEA parser. Update given GPSData structure with values from a
given NMEA sentence.
    /// Function does not check if a sentence is valid and has a good
checksum.
    /// This should be done earlier.
    ///
    /// Returns true if sentence was parsed succesfully.
    /// false if sentence was invalid, thrown exception or is not
implemented by this parser.
    public static bool UpdateGPSData(string sentence, GPSData currentdata)
    {
        // decode NMEA sentence and update fields of GPS Data accordingly
        currentdata.Protocol = GPSData.Protocols.NMEA;

        string cmd = ParseCommand(sentence);
        string[] cmdparams = ParseParameters(sentence);

        try
        {
            if (cmd == "GPRMC")
            {
                currentdata.NavigationStatus = cmdparams[1];// Status, V =
Navigation receiver warning
                    currentdata.Latitude = GetDoubleValue(cmdparams[2]);
                    currentdata.LatitudeNS = cmdparams[3][0];
                    currentdata.Longitude = GetDoubleValue(cmdparams[4]);

            *******  etc etc ****
            }
            else if (cmd == "GPRMB")
            {
            }
            else if (cmd == "GPGGA")
            {
            }
            else if (cmd == "GPGSA")
            {
            }
            Else, else, else --- other NMEA sentences
              "GPGSV","PGRME","GPGLL", "PGRMZ", "PGRMM", "GPBOD",
 "GPRTE"
```

```
            currentdata.HistoryGPSSentencesProcessed++;
        }
    }
```