ZylGPSReceiver 3.85

ZylGPSReceiver is a Delphi & C++Builder component collection that communicates with a GPS receiver (Global Positioning System).
It returns latitude, longitude, altitude, speed, heading and many other useful parameters of the current position and the parameters of the satellites in view. The component is extended to calculate distances and make conversions between different measurement units.
This component works with any NMEA 0183 compilant GPS receiver connected to one of the serial ports. NMEA 0183 (or NMEA for short) is a combined electrical and data specification for communication between marine electronic devices such as echo sounder, sonars, Anemometer (winds speed and direction), gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the US-based National Marine Electronics Association.
You can use it also with USB, IrDA and Bluetooth devices, because these devices have a driver that redirects the input from the USB, IrDA or Bluetooth port to a virtual serial port (you can check it in System/Device Manager/Ports). If your USB device is not provided with such a driver, then use a USB controller whose vendor provides a virtual serial port driver, such as FTDI or use a USB/RS-232 adapter. For Garmin receivers you have to install Spanner software.

The demo version is fully functional in Delphi and C++Builder IDE, but it displays a nag dialog (the licensed version will, of course, not have a nag dialog and will not be limited to the IDE). The package includes demo programs for Delphi and C++Builder and a help file with the description of the component.

Supported Operating Systems: Windows 2000/XP/Serv2003/Vista/Serv2008/7/8/Serv2012/10
Available for: Delphi 11.0 Alexandria (Win32 & Win64), Delphi 10.4 Sydney (Win32 & Win64), Delphi 10.3 Rio (Win32 & Win64), Delphi 10.2 Tokyo (Win32 & Win64), Delphi 10.1 Berlin (Win32 & Win64), Delphi 10 Seattle (Win32 & Win64), Delphi XE8 (Win32 & Win64), Delphi XE7 (Win32 & Win64), Delphi XE6 (Win32 & Win64), Delphi XE5 (Win32 & Win64), Delphi XE4 (Win32 & Win64), Delphi XE3 (Win32 & Win64), Delphi XE2 (Win32 & Win64), Delphi XE, Delphi 2010, Delphi 2009, Delphi 2007, Delphi 2006, Delphi 7, Delphi 6, Delphi 5, C++Builder 11.0 Alexandria (Win32 & Win64), C++Builder 10.4 Sydney (Win32 & Win64), C++Builder 10.3 (Win32 & Win64), C++Builder 10.2 (Win32 & Win64), C++Builder 10.1 (Win32 & Win64), C++Builder 10 (Win32 & Win64), C++Builder XE8 (Win32 & Win64), C++Builder XE7, C++Builder XE6, C++Builder XE5, C++Builder XE4, C++Builder XE3, C++Builder XE2, C++Builder XE, C++Builder 2010, C++Builder 2009, C++Builder 2007, C++Builder 2006, C++Builder 6, Turbo Delphi, Turbo C++
Remarks:
- The Delphi 2006 version is fully compatible with Turbo Delphi
- The C++Builder 2006 version is fully compatible with Turbo C++

Insatallation:
If you have a previous version of the component installed, you must remove it completely before installing this version. To remove a previous installation, proceed as follows:
-Start the IDE, open the packages page by selecting Component - Install Packages
-Select ZylGPSRecPack package in the list and click the Remove button
-Open Tools - Environment Options - Library and remove the library path pointing to ZylGPSReceiver folder
-Close the IDE
-Browse to the folder where your bpl and dcp files are located (default is $(DELPHI)\Projects\Bpl for Delphi, $(BCB)\Projects\Bpl for C++ Builder).

-Delete all of the files related to ZylGPSReceiver
-Delete or rename the top folder where ZylGPSReceiver is installed
-Start regedit (click Start - Run, type "regedit.exe" and hit Enter). Open the key HKEY_CURRENT_USER\Software\Borland\<compiler>\<version>\Palette and delete all name/value items in the list related to ZylGPSReceiver. (<compiler> is either "Delphi" or "C++Builder", <version> is the IDE version you have installed)

-Unzip the zip file and open the ZylGPSRecPack.dpk file in Delphi (ZylSerialPortPack.bpk or ZylSerialPortPack.cbproj file in C++Builder), compile and install it
and add to Tools/Environment Options/Library (in older Delphi/C++Builder menu) or Tools/Options/Delphi Options/Library/Library Path (in newer Delhi menu) or Tools/Options/C++ Options/Paths and Directories/Library Path & Include Path (in newer C++Builder menu, in C++Builder 10 or later, set them also for the classic compiler) the path of the installation (where the ZylGPSReceiver.dcu file is located). The component will be added to the "Zyl Soft" tab of the component palette. After you have the component on your component palette, you can drag and drop it to any form, where you can set its properties by the Object Inspector and you can write event handlers selecting the Events tab of the Object Inspector and double clicking the preferred event.
If you still have problems in C++Builder, running an application, which contains the component, then open the project and in C++Builder menu, Project/Options/Packages and uncheck "Build with runtime packages".
-another possible problem with C++Builder: Go to Project options, C++ Linker, and uncheck Link with dynamic RTL.

-It is indicated to use this component with "Stop on Delphi exception" option deactivated. You can do this from Delphi / C++Builder menu, "Tools/Debugger Options/Language Exceptions/Stop on Delphi exceptions" in older versions or Tools/Options/Debugger Options/Embarcadero Debuggers/Language Exceptions/Notify on language exceptions in newer versions, otherwise you will have a break at all the handled exceptions.

64-bit platform:
Delphi/C++Builder 64-bit support is only for runtime, so you have to use it

in the folllwing way:
Install the 32-bit version of the component as it described above and add to Tools/Options/Delphi Options/Library/Library Path, selected platform: 64-bit Windows the path of the Win64 subfolder of the component.
Before compiling the host application for 64-bit Windows, right click on Target Platforms, Add Platform and add 64-bit Windows (Make the selected platform active). If you compile the application in this way, it will be a native 64-bit application.

Constants:

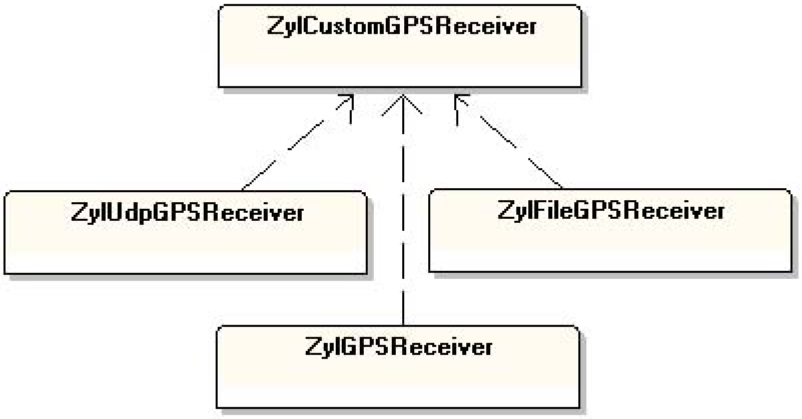
RADIUS_EARTH = 6378.14

Types:

TCardinalPoint = (cpNorth, cpSouth, cpEast, cpWest);
TDirection = (dirForward, dirLeft, dirRight);
TNMEACommands = set of (GPAAM, GPBWC, GPGGA, GPGLL, GPMSS, GPRMB, GPRMC, GPGSA, GPGSV, GPVTG, GPZDA, GPWPL, GPRTE, GPXTE, GPHDT, GPHDM, GPHDG, AllNMEA);
TCommPort = (spNone, spCOM1, spCOM2, spCOM3, spCOM4, spCOM5, spCOM6, spCOM7, spCOM8, spCOM9, spCOM10, spCOM11, spCOM12, spCOM13, spCOM14, spCOM15, spCOM16, spCOM17, spCOM18, spCOM19, spCOM20,
spCOM21, spCOM22, spCOM23, spCOM24, spCOM25, spCOM26, spCOM27, spCOM28, spCOM29, spCOM30,
spCOM31, spCOM32, spCOM33, spCOM34, spCOM35, spCOM36, spCOM37, spCOM38, spCOM39, spCOM40,
spCOM41, spCOM42, spCOM43, spCOM44, spCOM45, spCOM46, spCOM47, spCOM48, spCOM49, spCOM50,
spCOM51, spCOM52, spCOM53, spCOM54, spCOM55, spCOM56, spCOM57, spCOM58, spCOM59, spCOM60,
spCOM61, spCOM62, spCOM63, spCOM64, spCOM65, spCOM66, spCOM67, spCOM68, spCOM69, spCOM70,
spCOM71, spCOM72, spCOM73, spCOM74, spCOM75, spCOM76, spCOM77, spCOM78, spCOM79, spCOM80,
spCOM81, spCOM82, spCOM83, spCOM84, spCOM85, spCOM86, spCOM87, spCOM88, spCOM89, spCOM90,
spCOM91, spCOM92, spCOM93, spCOM94, spCOM95, spCOM96,

spCOM97, spCOM98, spCOM99, spCOM100,
spCOM101, spCOM102, spCOM103, spCOM104, spCOM105,
spCOM106, spCOM107, spCOM108, spCOM109, spCOM110,
spCOM111, spCOM112, spCOM113, spCOM114, spCOM115, spCOM116,
spCOM117, spCOM118, spCOM119, spCOM120,
spCOM121, spCOM122, spCOM123, spCOM124, spCOM125,
spCOM126, spCOM127, spCOM128, spCOM129, spCOM130,
spCOM131, spCOM132, spCOM133, spCOM134, spCOM135,
spCOM136, spCOM137, spCOM138, spCOM139, spCOM140,
spCOM141, spCOM142, spCOM143, spCOM144, spCOM145,
spCOM146, spCOM147, spCOM148, spCOM149, spCOM150,
spCOM151, spCOM152, spCOM153, spCOM154, spCOM155,
spCOM156, spCOM157, spCOM158, spCOM159, spCOM160,
spCOM161, spCOM162, spCOM163, spCOM164, spCOM165,
spCOM166, spCOM167, spCOM168, spCOM169, spCOM170,
spCOM171, spCOM172, spCOM173, spCOM174, spCOM175,
spCOM176, spCOM177, spCOM178, spCOM179, spCOM180,
spCOM181, spCOM182, spCOM183, spCOM184, spCOM185,
spCOM186, spCOM187, spCOM188, spCOM189, spCOM190,
spCOM191, spCOM192, spCOM193, spCOM194, spCOM195,
spCOM196, spCOM197, spCOM198, spCOM199, spCOM200,
spCOM201, spCOM202, spCOM203, spCOM204, spCOM205,
spCOM206, spCOM207, spCOM208, spCOM209, spCOM210,
spCOM211, spCOM212, spCOM213, spCOM214, spCOM215,
spCOM216, spCOM217, spCOM218, spCOM219, spCOM220,
spCOM221, spCOM222, spCOM223, spCOM224, spCOM225,
spCOM226, spCOM227, spCOM228, spCOM229, spCOM230,
spCOM231, spCOM232, spCOM233, spCOM234, spCOM235,
spCOM236, spCOM237, spCOM238, spCOM239, spCOM240,
spCOM241, spCOM242, spCOM243, spCOM244, spCOM245,
spCOM246, spCOM247, spCOM248, spCOM249, spCOM250,
spCOM251, spCOM252, spCOM253, spCOM254, spCOM255);
TCommPortSet = set of TCommPort;
TBaudRate = (br000075, br000110, br000134, br000150, br000300,
br000600, br001200, br001800,
br002400, br004800, br007200, br009600, br014400, br019200, br038400,
br057600,

br115200, br128000, br230400, br256000, br460800, br921600, brCustom);
TStopBits = (sb1Bit, sb1_5Bits, sb2Bits);
TDataWidth = (dw5Bits, dw6Bits, dw7Bits, dw8Bits);
TParityBits = (pbNone, pbOdd, pbEven, pbMark, pbSpace);
THwFlowControl = (hfNONE, hfDTRDTS, hfRTSCTS);
TSwFlowControl = (sfNONE, sfXONXOFF);
TArrivalEvent = procedure(Sender: TObject; const WayPoint: TWayPoint; const CircleRadius: Extended) of object;
TConnectEvent = procedure(Sender: TObject; Port: TCommPort) of object;
TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) of object;
TParamChangeEvent = procedure(Sender: TObject; Value: Extended) of object;
TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) of object;
TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) of object;
TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) of object;
TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) of object;
TDetectEvent = procedure(Sender: TObject; Port: TCommPort; BaudRate: TBaudRate; var Cancel: Boolean) of object;
TCommandEvent = procedure(Sender: TObject; Command: AnsiString) of object;
TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) of object;
EZylGPSReceiverException = class(Exception); //custom exception class
TSatelliteTypes = set of (AllSatellites, GpsSatellite, GlonassSatellite, BeidouSatellite, GalileoSatellite, QzssSatellite, IrnssSatellite, UnknownSatellite)



Buy Now!

http://www.zylsoft.com
info@zylsoft.com

ZylCustomGPSReceiver

TZylCustomGPSReceiver - Description:

ZylCustomGPSReceiver is a custom Delphi / C++Builder component, which is designed to be the base class of any kind of GPS receivers.
This component contains and NMEA decoder engine and it works with any NMEA compatible GPS receiver.
You can extend this class easily to process NMEA data from any kind of sources as files, sockets, web services and so on.

TZylCustomGPSReceiver - Properties:

ForceCheckSum - if this property is false the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored.
NMEALog - if this property is true a logfile will be created with any valid NMEA command.
NMEATerminator - NMEA sentence terminator.
NMEAPrefix - NMEA sentence prefix.
LogDateTime - if this property is true, the datetime will be added to every line of the log file.
LogDateTimeFormat: TFormatSettings - date-time format for logging.
LogFile - name and path of the log file.
WayPoints - contains the current waypoints.
ActiveRoute - the active route.
Track - the last registered track. The track is registered between Open and Close methods.
Commands - enable/disable NMEA sentences
Sentence Description:
GPGGA - Global positioning system fixed data
GPGLL - Geographic position - latitude / longitude
GPGSA - GNSS DOP and active satellites
GPGSV - GNSS satellites in view
GPRMC - Recommended minimum specific GNSS data
GPVTG - Course over ground and ground speed

GPMSS - Beacon Receiver Status
GPAAM - Waypoint Arrival Alarm
GPRMB - Recommended minimum navigation info
GPBWC - Bearing and distance to waypoint, great circle
GPWPL - Waypoint Location
GPRTE - Routes
GPXTE - Cross-Track Error, Measured
GPHDT - True Heading
GPHDM - Magnetic heading
GPHDG - Magnetic deviation and variation for calculating magnetic or true heading
GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences
SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.

TZylCustomGPSReceiver - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Base functionality methods:

procedure FeedGPSReceiver(strNMEA: AnsiString) - Use this method to feed the GPS receiver with NMEA data from the source you wish.

Conversion methods:

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS

(DegreeMinuteSecond) to DM (DegreeMinute)

function DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended): Extended- converts DMS (DegreeMinuteSecond) to DM (DegreeMinute - concatenated format: DDMM.mmmm)

function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians

procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)

function FeetToMeters(pDist: Extended): Extended - converts feet to meters

function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots

function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h

function KmToMiles(pDist: Extended): Extended - converts kilometers to miles

function MetersToFeet(pDist: Extended): Extended - converts meters to feet

function MilesToKm(pDist: Extended): Extended - converts miles to kilometers

function NauticalMilesToKm(pDist: Extended): Extended - converts nautical miles to km

function KmToNauticalMiles(pDist: Extended): Extended - converts km to nautical miles

procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values

procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values

procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values

procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction:

TCardinalPoint) - converts radians to DMS for longitude values
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

GPS Position related methods:

procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas
procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
The pen color indicates the satellite type:
GPS (USA) - White
Glonass (Russia) - Purple
Beidou (China) - Yellow
Galileo (Europe) - Blue
Qzs (Japan) - Green
Others - Silver
The fill color indicates the signal level:
SignalToNoiseRatio > 40 = Green
SignalToNoiseRatio > 25 = Lime
SignalToNoiseRatio > 10 = Yellow
SignalToNoiseRatio > 0 = Red
SignalToNoiseRatio = = Grey
function GetAltitude: Extended - returns altitude in meters
function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude

direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees. It's positive for east and negative for west.
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude direction
procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns longitude of the current position
function GetMagnetic_Variation: Extended - returns magnetic variation in degrees. It's positive for east and negative for west.
function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees
function GetMagnetic_Course: Extended - returns magnetic course in decimal degrees
function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D
function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix
function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined
function GetPDOP: Extended - returns PDOP (position dilution of precision)
function GetPosition: TGPSPosition - returns the parameters of the actual position
function GetRawData: AnsiString - returns the received raw NMEA data
function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver
function GetSatellites: TList - returns the list of satellites in view (objects

of TSatellite type)
function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count
function GetSpeed_KMH: Extended - returns speed in km/h
function GetSpeed_Knots: Extended - returns speed in knots
function GetUTCDateTime: TDateTime - returns UTC datetime
function GetVDOP: Extended - returns VDOP (vertical dilution of precision)
function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
function RouteToNMEA(const Route: TRoute): AnsiString - converts route to NMEA string
function ShowOnGoogleMaps(const zoom: Integer = 15): Integer - shows the current position on google maps (internet explorer)
function WayPointToNMEA(const WayPoint: TWayPoint): AnsiString - converts waypoint to NMEA string

Useful methods:
function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS
function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees
function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current

position; params in DMS
function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees
function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS
function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in km; params in decimal degrees
function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers
function DistanceTo_KM(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
function Distance_Miles(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute:

Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.
function GetAverageSatelliteSnr(): Extended - returns the average SignalToNoiseRatio of satellites using the highest 4 values.

TZylCustomGPSReceiver - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.
OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint: TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.
OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.
OnSend: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the sent data from the Buffer parameter.
OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.
OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.
OnAltitudeChange: TParamChangeEvent = procedure(Sender:

TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.

OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees.

OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.

OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.

OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.

OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.

OnUnknownCommand: TUnknownCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.

OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

TZylGPSReceiver

TZylSerialGPSReceiver = TZylGPSReceiver;

TDeviceSetting = class
property CommPort: TCommPort read FCommPort write FCommPort;
property BaudRate: TBaudRate read FBaudRate write FBaudRate;
constructor Create(); overload;
constructor Create(commPort: TCommPort; baudRate: TBaudRate);
overload;
end;

TDeviceSettings = class
property Count: Integer read GetCount;
property Items[Index: Integer]: TDeviceSetting read GetItem write SetItem;
default;

constructor Create();
destructor Destroy; override;
function Add(const Item: TDeviceSetting): Integer;
procedure Delete(Idx: Integer);
procedure Clear;
end;

TZylGPSReceiver - Description:

ZylGPSReceiver is a Delphi / C++Builder component that communicates with a serial GPS receiver.
This component works with any NMEA compatible receiver connected to one of the serial ports.
You can use it also with USB devices, because these devices usually have a driver that redirects the input from the USB port to a virtual serial port. If your device is not provided with such a driver, then use a USB controller whose vendor provides a virtual serial port driver, such as FTDI or use a USB/RS-232 adapter.

With this component you will be able to develop robust GPS Delphi or C++Builder applications.

TZylGPSReceiver - Properties:

Port - the serial port where the receiver is connected
BaudRate - baud rate of the serial port
CustomPortName: AnsiString - custom port name, when Port property is set to spCustom. Now you can open ports with any name.
CustomBaudRate - baud rate value at which the communication device operates, when BaudRate property is set to brCustom
DataWidth - number of bits in the bytes transmitted and received
StopBits - number of stop bits to be used
Parity - parity scheme to be used
HwFlowControl: THwFlowControl - hardware flow control
SwFlowControl: TSwFlowControl - software flow control
EnableDTROnOpen - enable / disable DTR when the port is open
EnableRTSOnOpen - enable / disable RTS when the port is open
Priority - priority of the receiver thread
Delay - Time interval between two receivings in milliseconds
AutoReconnect: Boolean - Set this property to true, if you want to automatically reconnect to the serial port after a OnFault event, when the port is available again. Set AutoReconnect to true, before the port is faulted, otherwise it will have no effect.
The default value is false.
AutoReconnectCheckInterval: Integer - The time interval in milliseconds the serial port is trying to periodically reconnect, after a OnFault event occur, if AutoReconnect is set to true. It must be a positive value. The default value is 4000.
CloseWhenLineStatusIsZero: Boolean - when this property is true and line status is empty, the port will be closed automatically, if line status was not empty, after you opened the port.
ConnectionTimeout (milliseconds) - you can set a time-out value to automatically close the connection and fire the OnTimeout event, if there is no data received several milliseconds. A value of zero indicates that time-out is not used.
IdleInterval (milliseconds) - you can set an idle interval value to automatically to fire the OnIdle event, if there is no data received several

seconds. A value of zero indicates that idle is not used.

IsIdle - true, when the connection is in idle state.

ForceCheckSum - if this property is false the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored.

NeedSynchronization: Boolean - set this property to true for thread safety. If you use the component in ActiveX environment, set this property to false. The default value is true.

NMEALog - if this property is true a logfile will be created with any valid NMEA command.

NMEATerminator - NMEA sentence terminator.

NMEAPrefix - NMEA sentence prefix.

LogDateTime - if this property is true, the datetime will be added to every line of the log file.

LogDateTimeFormat: TFormatSettings - date-time format for logging.

LogFile - name and path of the log file.

WayPoints - contains the current waypoints

ActiveRoute - the active route.

Commands - enable/disable NMEA sentences.

Sentence Description:

GPGGA - Global positioning system fixed data

GPGLL - Geographic position - latitude / longitude

GPGSA - GNSS DOP and active satellites

GPGSV - GNSS satellites in view

GPRMC - Recommended minimum specific GNSS data

GPVTG - Course over ground and ground speed

GPMSS - Beacon Receiver Status

GPAAM - Waypoint Arrival Alarm

GPRMB - Recommended minimum navigation info

GPBWC - Bearing and distance to waypoint, great circle

GPWPL - Waypoint Location

GPRTE - Routes

GPXTE - Cross-Track Error, measured

GPHDT - True Heading

GPHDM - Magnetic heading

GPHDG - Magnetic deviation and variation for calculating magnetic or true heading

GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences
SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.
IsFaulted: Boolean - indicates that the last connection was faulted.

TZylGPSReceiver - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Communication methods:

procedure Close - stops communication
function DetectGPS(var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first available (not busy/open) GPS receiver connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rates values will be checked.
function DetectGPS(startBaudRate, endBaudRate: TBaudRate; var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first available (not busy/open) GPS receiver connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rate values between startBaudRate and endBaudRate will be checked.
function DetectGPS(startPort: TCommPort; const startBaudRate, endBaudRate: TBaudRate; var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first available (not busy/open) GPS receiver after startPort, connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rate values between startBaudRate and endBaudRate will be checked.
function DetectGPS(const startBaudRate, endBaudRate: TBaudRate; var pPort: String; var pBaudRate: TBaudRate): Boolean - detects the

first available (not busy/open) GPS receiver connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rate values between startBaudRate and endBaudRate will be checked. It works for serial ports with different naming convention.

function DetectGPS(var pPort: String; var pBaudRate: TBaudRate): Boolean - detects the first available (not busy/open) GPS receiver connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rates values will be checked. It works for serial ports with different naming convention.

function FastDetectGPS(var pPort: TCommPort; var pBaudRate: TBaudRate): Boolean - detects the first available (not busy/open) GPS receiver connected to the system, returns as output parameters the communication port and baud rate. Only standard baud rate values between 4800 and 115200 will be checked.

function DetectAllGPS(startPort: TCommPort; const startBaudRate, endBaudRate: TBaudRate): TDeviceSettings - detects all the GPS receivers connected to the system. Only ports starting with startPort and baud rate values between startBaudRate and endBaudRate will be checked.

function DetectAllGPS(): TDeviceSettings - detects all the GPS receivers connected to the system.

function FastDetectAllGPS(): TDeviceSettings - detects all the GPS receivers connected to the system. Only baud rate values higher than 4800 will be checked.

function GetExistingCommPorts: TCommPortSet - returns the existing serial ports of the system

function IsConnected: TCommPort - returns the comm port where gps is connected to.

function IsExistingCommPort(pport: String): Boolean - return true if the pport parameter contains an existing serial port, otherwise false.

procedure Open - starts communication**.**

function Send(str: AnsiString): DWORD - sends a string to the gps receiver and returns the number of bytes successfully sent**.**

procedure ResetIdleState - resets the idle state of the port

Conversion methods:

function BaudRateToInt(pBaudRate: TBaudRate): Integer - converts TBaudRate type to integer

function CommPortToString(Port: TCommPort): AnsiString - converts TCommPort to String
function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)
function TZylCustomGPSReceiver.DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended): Extended- converts DMS (DegreeMinuteSecond) to DM (DegreeMinute - concatenated format: DDMM.mmmm)
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)
function FeetToMeters(pDist: Extended): Extended - converts feet to meters
function IntToBaudRate(Value: Integer): TBaudRate - converts integer to TBaudRate type
function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots
function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h
function KmToMiles(pDist: Extended): Extended - converts kilometers to miles
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended;

var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values
function MetersToFeet(pDist: Extended): Extended - converts meters to feet
function MilesToKm(pDist: Extended): Extended - converts miles to kilometers
function NauticalMilesToKm(pDist: Extended): Extended - converts nautical miles to km
function KmToNauticalMiles(pDist: Extended): Extended - converts km to nautical miles
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees
function StringToCommPort(Port: AnsiString): TCommPort - converts String to TCommPort

GPS Position related methods:

procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas
procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
The pen color indicates the satellite type:
GPS (USA) - White
Glonass (Russia) - Purple
Beidou (China) - Yellow
Galileo (Europe) - Blue
Qzs (Japan) - Green
Others - Silver
The fill color indicates the signal level:
SignalToNoiseRatio > 40 = Green
SignalToNoiseRatio > 25 = Lime
SignalToNoiseRatio > 10 = Yellow
SignalToNoiseRatio > 0 = Red
SignalToNoiseRatio = = Grey
function GetAltitude: Extended - returns altitude in meters

function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude direction
procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns longitude of the current position
function GetMagnetic_Variation: Extended - returns magnetic variation in degrees.It's positive for east and negative for west.
function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees
function GetMagnetic_Course: Extended - returns magnetic course in

decimal degrees
function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D
function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix
function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined
function GetPDOP: Extended - returns PDOP (position dilution of precision)
function GetPosition: TGPSPosition - returns the parameters of the actual position
function GetRawData: AnsiString - returns the received raw NMEA data
function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver
function GetSatellites: TList - returns the list of satellites in view (objects of TSatellite type)
function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count
function GetSpeed_KMH: Extended - returns speed in km/h
function GetSpeed_Knots: Extended - returns speed in knots
function GetUTCDateTime: TDateTime - returns UTC datetime
function GetVDOP: Extended - returns VDOP (vertical dilution of precision)
function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
function RouteToNMEA(const Route: TRoute): AnsiString - converts route to NMEA string
procedure UploadRoute(const Route: TRoute) - uploads a route
procedure UploadWaypoint(const WayPoint: TWayPoint) - uploads a waypoint
function WayPointToNMEA(const WayPoint: TWayPoint): AnsiString - converts waypoint to NMEA string

Useful methods:
function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1:

Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS

function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees

function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in DMS

function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees

function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS

function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in km; params in decimal degrees

function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers

function DistanceTo_KM(Latitude, Longitude: Extended): Extended;

overload - returns distance between a point and the current position in miles; params in decimal degrees
function Distance_Miles(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.

TZylGPSReceiver - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.
OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint: TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.
OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.

OnSend: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the sent data from the Buffer parameter.

OnConnect: TConnectEvent = procedure(Sender: TObject; Port: TCommPort) - fires after a new connection was established. The Port parameter contains the serial port where the GPS receiver is connected to.

OnDisconnect: TConnectEvent = procedure(Sender: TObject; Port: TCommPort) - fires before a disconnection. The Port parameter contains the serial port where the GPS receiver was connected to.

OnFault: TConnectEvent==procedure(Sender: TObject; Port: TCommPort) - occures when the serial port communication is faulted. E.g.: when you unplug an USB device which communicates on a virtual serial port.

OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.

OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.

OnAltitudeChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.

OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters

contain the current latitude and longitude values in decimal degrees.
OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.
OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.
OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.
OnTimeout: TNotifyEvent - fires when there is no data received several milliseconds, the interval is specified in the ConnectionTimeout property.
OnIdle: TNotifyEvent - fires when there is no data received several milliseconds, the interval is specified in the IdleInterval property.
OnResume: TNotifyEvent - Occurs when the receiver is idle and data is received.
OnDetect: TDetectEvent = procedure(Sender: TObject; Port: TCommPort; BaudRate: TBaudRate; var Cancel: Boolean) - fires when the GPS receiver detection is in progress and there are new values of port or baud rate in view. Port and BaudRate parameters represent the current port and baud rate values used in the detection. If you set the Cancel parameter to true, then the detection will be cancelled.
OnReconnect: TReconnectEvent=procedure(Sender: TObject; const Port: TCommPort; const BaudRate: TBaudRate; var Cancel: Boolean) - occurs when the serial port is trying to reconnect, after fault. AutoReconnect must be true. If you set the Cancel parameter to true, then the reconnection will be cancelled.
OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.
OnUnknownCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in

the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.
OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

http://www.zylsoft.com
info@zylsoft.com

ZylUdpGPSReceiver (ZylUdp10GPSReceiver)

ZylUdpGPSReceiver - Description It's an extension of ZylCustomGPSReceiver, which processes NMEA data from an UDP socket.
ZylUdpGPSReceiver uses Indy 9 socket components (http://www.indyproject.org).
ZylUdp10GPSReceiver uses Indy 10 socket components (http://www.indyproject.org).
You can find the full source code of these components in the download package.

TZylUdpGPSReceiver - Properties:

Port - the UDP port where the receiver is connected
RemoteHost - the IP of the remote computer
ForceCheckSum - if this property is false the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored.
NMEALog - if this property is true a logfile will be created with any valid NMEA command.
NMEATerminator - NMEA sentence terminator.
NMEAPrefix - NMEA sentence prefix.
LogDateTime - if this property is true, the datetime will be added to every line of the log file.
LogDateTimeFormat: TFormatSettings - date-time format for logging.
LogFile - name and path of the log file.
WayPoints - contains the current waypoints.
ActiveRoute - the active route.
Commands - enable/disable NMEA sentences
Sentence Description:
GPGGA - Global positioning system fixed data
GPGLL - Geographic position - latitude / longitude
GPGSA - GNSS DOP and active satellites
GPGSV - GNSS satellites in view

GPRMC - Recommended minimum specific GNSS data
GPVTG - Course over ground and ground speed
GPMSS - Beacon Receiver Status
GPAAM - Waypoint Arrival Alarm
GPRMB - Recommended minimum navigation info
GPBWC - Bearing and distance to waypoint, great circle
GPWPL - Waypoint Location
GPRTE - Routes
GPXTE - Cross-Track Error, Measured
GPHDT - True Heading
GPHDM - Magnetic heading
GPHDG - Magnetic deviation and variation for calculating magnetic or true heading
GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences
SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.

TZylUdpGPSReceiver - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Communication methods:

procedure Close - stops communication
procedure Open - starts communication

Conversion methods:

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees

procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)
function FeetToMeters(pDist: Extended): Extended - converts feet to meters
function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots
function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h
function KmToMiles(pDist: Extended): Extended - converts kilometers to miles
function MetersToFeet(pDist: Extended): Extended - converts meters to feet
function MilesToKm(pDist: Extended): Extended - converts miles to kilometers
function NauticalMilesToKm(pDist: Extended): Extended - converts nautical miles to km
function KmToNauticalMiles(pDist: Extended): Extended - converts km to nautical miles
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values

function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

GPS Position related methods:

procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas
procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
function GetAltitude: Extended - returns altitude in meters
function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude

direction

procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns longitude of the current position

function GetMagnetic_Variation: Extended - returns magnetic variation in degrees. It's positive for east and negative for west.

function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees

function GetMagnetic_Course: Extended - returns magnetic course in decimal degrees

function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D

function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix

function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined

function GetPDOP: Extended - returns PDOP (position dilution of precision)

function GetPosition: TGPSPosition - returns the parameters of the actual position

function GetRawData: AnsiString - returns the received raw NMEA data

function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver

function GetSatellites: TList - returns the list of satellites in view (objects of TSatellite type)

function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count

function GetSpeed_KMH: Extended - returns speed in km/h

function GetSpeed_Knots: Extended - returns speed in knots

function GetUTCDateTime: TDateTime - returns UTC datetime

function GetVDOP: Extended - returns VDOP (vertical dilution of precision)

function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS

function RouteToNMEA(const Route: TRoute): AnsiString - converts route to NMEA string

procedure UploadRoute(const Route: TRoute) - uploads a route

procedure UploadWaypoint(const WayPoint: TWayPoint) - uploads a waypoint
function WayPointToNMEA(const WayPoint: TWayPoint): AnsiString - converts waypoint to NMEA string

Useful methods:
function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS
function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees
function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in DMS
function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees
function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS

function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in km; params in decimal degrees
function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers
function DistanceTo_KM(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
function Distance_Miles(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees.
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.

TZylUdpGPSReceiver - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.

OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint: TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.

OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.

OnSend: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the sent data from the Buffer parameter.

OnConnect: TConnectEvent = procedure(Sender: TObject; Port: Integer) - fires after a new connection was established. The Port parameter contains the UDP port where the GPS receiver is connected to.

OnDisconnect: TConnectEvent = procedure(Sender: TObject; Port: Integer) - fires before a disconnection. The Port parameter contains the UDP port where the GPS receiver was connected to.

OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.

OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.

OnAltitudeChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender:

TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.
OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees.
OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.
OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.
OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.
OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.
OnUnknownCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.
OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

http://www.zylsoft.com
info@zylsoft.com

Zyl soft

ZylFileGPSReceiver

ZylFileGPSReceiver - Description

It's an extension of ZylCustomGPSReceiver, which processes NMEA data from a text file. You can find the full source code of this component in the download package.

TZylFileGPSReceiver - Properties:

FileName - the name of the file where the receiver is connected
Delay - time interval between two receivings in milliseconds
ForceCheckSum - if this property is false, the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored
IsConnected - if this property is true, the receiver is connected to a serial port
IsPaused - if this property is true, the receiver is paused
NMEALog - if this property is true a logfile will be created with any valid NMEA command.
NMEATerminator - NMEA sentence terminator.
NMEAPrefix - NMEA sentence prefix.
LogDateTime - if this property is true, the datetime will be added to every line of the log file.
LogDateTimeFormat: TFormatSettings - date-time format for logging.
LogFile - name and path of the log file.
WayPoints - contains the current waypoints.
ActiveRoute - the active route.
Commands - enable/disable NMEA sentences
Sentence Description:
GPGGA - Global positioning system fixed data
GPGLL - Geographic position - latitude / longitude
GPGSA - GNSS DOP and active satellites
GPGSV - GNSS satellites in view
GPRMC - Recommended minimum specific GNSS data
GPVTG - Course over ground and ground speed

GPMSS - Beacon Receiver Status
GPAAM - Waypoint Arrival Alarm
GPRMB - Recommended minimum navigation info
GPBWC - Bearing and distance to waypoint, great circle
GPWPL - Waypoint Location
GPRTE - Routes
GPXTE - Cross-Track Error, measured
GPHDT - True Heading
GPHDM - Magnetic heading
GPHDG - Magnetic deviation and variation for calculating magnetic or true heading
GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences
SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.

TZylFileGPSReceiver - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Communication methods:

procedure Close - stops the communication
procedure Continue - continues the communication, after it was paused
procedure Open - starts the communication
procedure Pause - pauses the communication

Conversion methods:

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)

function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)
function FeetToMeters(pDist: Extended): Extended - converts feet to meters
function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots
function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h
function KmToMiles(pDist: Extended): Extended - converts kilometers to miles
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values
function MetersToFeet(pDist: Extended): Extended - converts meters to feet
function MilesToKm(pDist: Extended): Extended - converts miles to kilometers
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

GPS Position related methods:

procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas
procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
function GetAltitude: Extended - returns altitude in meters
function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude direction
procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) -

returns longitude of the current position
function GetMagnetic_Variation: Extended - returns magnetic variation in degrees. It's positive for east and negative for west..
function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees
function GetMagnetic_Course: Extended - returns magnetic course in decimal degrees
function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D
function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix
function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined
function GetPDOP: Extended - returns PDOP (position dilution of precision)
function GetPosition: TGPSPosition - returns the parameters of the actual position
function GetRawData: AnsiString - returns the received raw NMEA data
function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver
function GetSatellites: TList - returns the list of satellites in view (objects of TSatellite type)
function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count
function GetSpeed_KMH: Extended - returns speed in km/h
function GetSpeed_Knots: Extended - returns speed in knots
function GetUTCDateTime: TDateTime - returns UTC datetime
function GetVDOP: Extended - returns VDOP (vertical dilution of precision)
function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
function RouteToNMEA(strRoute: AnsiString): AnsiString - converts route to NMEA data
function WayPointToNMEA(strName: AnsiString; dLatitude, dLongitude: Extended): AnsiString - converts waypoint in NMEA data. The parameters dLatitude and dLongitude are the coordonates of the waypoint in decimal degrees.

function WayPointToNMEA(strName: AnsiString; LatDeg, LatMin: Integer; LatSec: Extended; LatDir: TCardinalPoint; LongDeg, LongMin: Integer; LongSec: Extended; LongDir: TCardinalPoint): AnsiString - converts waypoint in NMEA data.

Useful methods:
function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS
function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees
function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in DMS
function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees
function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS

function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in km; params in decimal degrees
function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers
function DistanceTo_KM(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
function Distance_Miles(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees.
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.

TZylFileGPSReceiver - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.

OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint: TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.

OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.

OnConnect: TConnectEvent = procedure(Sender: TObject; FileName: String) - fires after a new connection was established. The The FileName parameter contains the name of the file where the GPS receiver is connected to.

OnDisconnect: TConnectEvent = procedure(Sender: TObject; FileName: String) - fires before a disconnection. The FileName parameter contains the name of the file where the GPS receiver was connected to.

OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.

OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.

OnAltitudeChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.

OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees.
OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.
OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.
OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.
OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.
OnUnknownCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.
OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

http://www.zylsoft.com
info@zylsoft.com

Zyl soft

ZylGpxGPSReceiver

ZylGpxGPSReceiver - Description It's an extension of ZylCustomGPSReceiver, which processes data from GPX files.

ZylGpxGPSReceiver - Properties:

FileName - the name of the file where the receiver is connected
Delay - time interval between two receivings in milliseconds
ForceCheckSum - if this property is false, the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored
IsConnected - if this property is true, the receiver is connected to a serial port
IsPaused - if this property is true, the receiver is paused
NMEALog - if this property is true a logfile will be created with any valid NMEA command.
NMEATerminator - NMEA sentence terminator.
NMEAPrefix - NMEA sentence prefix.
LogDateTime - if this property is true, the datetime will be added to every line of the log file.
LogDateTimeFormat: TFormatSettings - date-time format for logging.
LogFile - name and path of the log file.
WayPoints - contains the current waypoints.
ActiveRoute - the active route.
Commands - enable/disable NMEA sentences
Sentence Description:
GPGGA - Global positioning system fixed data
GPGLL - Geographic position - latitude / longitude
GPGSA - GNSS DOP and active satellites
GPGSV - GNSS satellites in view
GPRMC - Recommended minimum specific GNSS data
GPVTG - Course over ground and ground speed
GPMSS - Beacon Receiver Status
GPAAM - Waypoint Arrival Alarm

GPRMB - Recommended minimum navigation info
GPBWC - Bearing and distance to waypoint, great circle
GPWPL - Waypoint Location
GPRTE - Routes
GPXTE - Cross-Track Error, measured
GPHDT - True Heading
GPHDM - Magnetic heading
GPHDG - Magnetic deviation and variation for calculating magnetic or true heading
GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences

SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.

ZylGpxGPSReceiver - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Communication methods:

procedure Close - stops the communication
procedure Continue - continues the communication, after it was paused
procedure Open - starts the communication
procedure Pause - pauses the communication

Conversion methods:

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS

(DegreeMinuteSecond) to decimal degrees
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)
function FeetToMeters(pDist: Extended): Extended - converts feet to meters
function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots
function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h
function KmToMiles(pDist: Extended): Extended - converts kilometers to miles
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values
function MetersToFeet(pDist: Extended): Extended - converts meters to feet
function MilesToKm(pDist: Extended): Extended - converts miles to kilometers
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

GPS Position related methods:
procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas

procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
function GetAltitude: Extended - returns altitude in meters
function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude direction
procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns longitude of the current position
function GetMagnetic_Variation: Extended - returns magnetic variation in degrees. It's positive for east and negative for west..

function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees
function GetMagnetic_Course: Extended - returns magnetic course in decimal degrees
function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D
function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix
function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined
function GetPDOP: Extended - returns PDOP (position dilution of precision)
function GetPosition: TGPSPosition - returns the parameters of the actual position
function GetRawData: AnsiString - returns the received raw NMEA data
function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver
function GetSatellites: TList - returns the list of satellites in view (objects of TSatellite type)
function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count
function GetSpeed_KMH: Extended - returns speed in km/h
function GetSpeed_Knots: Extended - returns speed in knots
function GetUTCDateTime: TDateTime - returns UTC datetime
function GetVDOP: Extended - returns VDOP (vertical dilution of precision)
function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
function RouteToNMEA(strRoute: AnsiString): AnsiString - converts route to NMEA data
function WayPointToNMEA(strName: AnsiString; dLatitude, dLongitude: Extended): AnsiString - converts waypoint in NMEA data. The parameters dLatitude and dLongitude are the coordonates of the waypoint in decimal degrees.
function WayPointToNMEA(strName: AnsiString; LatDeg, LatMin: Integer; LatSec: Extended; LatDir: TCardinalPoint; LongDeg,

LongMin: Integer; LongSec: Extended; LongDir: TCardinalPoint): AnsiString - converts waypoint in NMEA data.

Useful methods:

function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS
function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees
function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in DMS
function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees
function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS
function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in

km; params in decimal degrees
function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers
function DistanceTo_KM(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees
function Distance_Miles(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees.
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.

ZylGpxGPSReceiver - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.
OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint:

TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.

OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.

OnConnect: TConnectEvent = procedure(Sender: TObject; FileName: String) - fires after a new connection was established. The The FileName parameter contains the name of the file where the GPS receiver is connected to.

OnDisconnect: TConnectEvent = procedure(Sender: TObject; FileName: String) - fires before a disconnection. The FileName parameter contains the name of the file where the GPS receiver was connected to.

OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.

OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.

OnAltitudeChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.

OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees.

OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.

OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.

OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.

OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.

OnUnknownCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.

OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

http://www.zylsoft.com
info@zylsoft.com

ZylNMEADecoder

ZylNMEADecoder - Description Implements an NMEA decoder.
It works like a GPS Receiver, but you have to feed it with NMEA sentences programatically from the Decode method.

ZylNMEADecoder - Properties:

ForceCheckSum - if this property is false the checksum of any command is ignored, otherwise all the sentences which checksum is not correct will be ignored.

NMEALog - if this property is true a logfile will be created with any valid NMEA command.

NMEATerminator - NMEA sentence terminator.

NMEAPrefix - NMEA sentence prefix.

LogDateTime - if this property is true, the datetime will be added to every line of the log file.

LogDateTimeFormat: TFormatSettings - date-time format for logging.

LogFile - name and path of the log file.

WayPoints - contains the current waypoints.

ActiveRoute - the active route.

Commands - enable/disable NMEA sentences

Sentence Description:
GPGGA - Global positioning system fixed data
GPGLL - Geographic position - latitude / longitude
GPGSA - GNSS DOP and active satellites
GPGSV - GNSS satellites in view
GPRMC - Recommended minimum specific GNSS data
GPVTG - Course over ground and ground speed
GPMSS - Beacon Receiver Status
GPAAM - Waypoint Arrival Alarm
GPRMB - Recommended minimum navigation info
GPBWC - Bearing and distance to waypoint, great circle
GPWPL - Waypoint Location
GPRTE - Routes

GPXTE - Cross-Track Error, measured
GPHDT - True Heading
GPHDM - Magnetic heading
GPHDG - Magnetic deviation and variation for calculating magnetic or true heading
GPGNS - Fixes data for single or combined (GPS, GLONASS, possible future satellite systems, and systems combining these) satellite navigation systems
AllNMEA - All NMEA sentences
SatelliteTypes: TSatelliteTypes - Accepted satellite types. Do not include AllSatellites, if you want to accept only individual type of satellites. If you want to use in mixed mode, using more type of satellites, allways set SatelliteTypes to AllSatellites.

ZylNMEADecoder - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Communication methods:

procedure Decode(nmea: AnsiString) - feeds the GPS receiver.

Conversion methods:

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)

function FeetToMeters(pDist: Extended): Extended - converts feet to meters
function KmhToKnots(pSpeed: Extended): Extended - converts km/h to knots
function KnotsToKmh(pSpeed: Extended): Extended - converts knots to km/h
function KmToMiles(pDist: Extended): Extended - converts kilometers to miles
function MetersToFeet(pDist: Extended): Extended - converts meters to feet
function MilesToKm(pDist: Extended): Extended - converts miles to kilometers
function NauticalMilesToKm(pDist: Extended): Extended - converts nautical miles to km
function KmToNauticalMiles(pDist: Extended): Extended - converts km to nautical miles
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

GPS Position related methods:
procedure DrawSky(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the sky on the canvas
procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor) - draws the satellites on the canvas
function GetAltitude: Extended - returns altitude in meters

function GetGeo_Height: Extended - returns the difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
function GetHDOP: Extended - returns HDOP (horizontal dilution of precision)
function GetHeading: Extended - returns true heading in decimal degrees
function GetCourse: Extended - returns true course in decimal degrees
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
procedure GetLatitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns latitude of the current position
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude direction
procedure GetLongitudeParams(var Degree: Integer; var Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - returns longitude of the current position
function GetMagnetic_Variation: Extended - returns magnetic variation in degrees. It's positive for east and negative for west.
function GetMagnetic_Heading: Extended - returns magnetic heading in decimal degrees
function GetMagnetic_Course: Extended - returns magnetic course in

decimal degrees
function GetMode1(): Word - returns returns 1 = Auto 2D/3D, 2 = Forced 2D/3D
function GetMode2(): Word - returns 1 = no fix, 2 = 2D fix, 3 = 3D fix
function GetNavigationInfo(): TNavigationInfo - returns navigation info if a destination waypoint is defined
function GetPDOP: Extended - returns PDOP (position dilution of precision)
function GetPosition: TGPSPosition - returns the parameters of the actual position
function GetRawData: AnsiString - returns the received raw NMEA data
function GetReceiverStatus(): TReceiverStatus - returns the status of the gps receiver
function GetSatellites: TList - returns the list of satellites in view (objects of TSatellite type)
function GetSatelliteCount: Integer - returns the count of the satellites used to determine the fix position. If you need the count of all satellites in view, use GetSatellites.Count instead instead of Satellite_Count
function GetSpeed_KMH: Extended - returns speed in km/h
function GetSpeed_Knots: Extended - returns speed in knots
function GetUTCDateTime: TDateTime - returns UTC datetime
function GetVDOP: Extended - returns VDOP (vertical dilution of precision)
function IsFix: Word - returns if the position is a fix; 0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS
function RouteToNMEA(const Route: TRoute): AnsiString - converts route to NMEA string
function WayPointToNMEA(const WayPoint: TWayPoint): AnsiString - converts waypoint to NMEA string

Useful methods:
**function Bearing(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint;
Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint;**

Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between two points in kilometers; params in DMS

function Bearing(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended - returns bearing (course) in decimal degrees between two points; params in decimal degrees

function BearingTo(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in DMS

function BearingTo(Latitude, Longitude: Extended): Extended - returns bearing (course) in decimal degrees between a point and the current position; params in decimal degrees

function Distance_KM(Latitude_Degree1, Latitude_Minute1: Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in kilometers; params in DMS

function Distance_KM(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in km; params in decimal degrees

function DistanceTo_KM(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in kilometers

function DistanceTo_KM(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees

function Distance_Miles(Latitude_Degree1, Latitude_Minute1:

Integer; Latitude_Second1: Extended; Latitude_Direction1: TCardinalPoint; Longitude_Degree1, Longitude_Minute1: Integer; Longitude_Second1: Extended; Longitude_Direction1: TCardinalPoint; Latitude_Degree2, Latitude_Minute2: Integer; Latitude_Second2: Extended; Latitude_Direction2: TCardinalPoint; Longitude_Degree2, Longitude_Minute2: Integer; Longitude_Second2: Extended; Longitude_Direction2: TCardinalPoint): Extended; overload - returns distance between two points in miles; params in DMS
function Distance_Miles(Latitude1, Longitude1, Latitude2, Longitude2: Extended): Extended; overload - returns distance between two points in miles; params in decimal degrees
function DistanceTo_Miles(Latitude_Degree, Latitude_Minute: Integer; Latitude_Second: Extended; Latitude_Direction: TCardinalPoint; Longitude_Degree, Longitude_Minute: Integer; Longitude_Second: Extended; Longitude_Direction: TCardinalPoint): Extended - returns distance between a point and the current position in miles; params in DMS
function DistanceTo_Miles(Latitude, Longitude: Extended): Extended; overload - returns distance between a point and the current position in miles; params in decimal degrees.
procedure ExportTrackToGpx(fileName: String) - exports the last track to GPX xml file.

ZylNMEADecoder - Events:

OnActiveRouteReceive: TNotifyEvent - fires when the current active route is received.
OnArrival: TArrivalEvent = procedure(Sender: TObject; Waypoint: TWayPoint; CircleRadius: Extended) - fires when a destination waypoint is defined and an arrival alarm is received. The Waypoint parameter contains the the destination waypoint, the CircleRadius parameter the size of the circle radius in nautical miles.
OnReceive: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the received data from the Buffer parameter.
OnSend: TSendReceiveEvent = procedure(Sender: TObject; Buffer: AnsiString) - fires when new data was received. You can get the sent data from the Buffer parameter.

OnLatitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when latitude has changed. Degree, Minute, Second and Direction parameters are the components of the current latitude.

OnLongitudeChange: TPositionChangeEvent = procedure(Sender: TObject; Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint) - fires when longitude has changed. Degree, Minute, Second and Direction parameters are the components of the current longitude.

OnAltitudeChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when altitude has changed**.** Value contains the current altitude in meters.

OnSpeedChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when speed has changed**.** Value contains the current speed in km/h.

OnHeadingChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when heading has changed**.** Value contains the current heading in degrees.

OnCourseChange: TParamChangeEvent = procedure(Sender: TObject; Value: Extended) - fires when course has changed**.** Value contains the current course in degrees.

OnPosition2DChange: TShortPositionChangeEvent = procedure(Sender: TObject; Latitude, Longitude: Extended) - fires when latitude or longitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees.

OnPosition3DChange: TShortPosition3DChangeEvent = procedure(Sender: TObject; Latitude, Longitude, Altitude: Extended) - fires when latitude or longitude or altitude has changed. Latitude and Longitude parameters contain the current latitude and longitude values in decimal degrees, Atlitude parameter contains the current altitude in meters.

OnSatelliteReceive: TSatelliteReceiveEvent = procedure(Sender: TObject; Satellite: TSatellite) - fires when one or more satellites are received. Satellite parameter contains a TSatellite object with the parameters of the received satellite.

OnSatellitesReceive: TNotifyEvent - fires when a complete list of satellites in view is received. You can get the list of satellites in view GetSatellites method.

OnNewCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a new valid NMEA sentence is received. The Command parameter contains the sentence.
OnUnknownCommand: TCommandEvent = procedure(Sender: TObject; Command: AnsiString) - fires when a valid, but unknown NMEA sentence is received. The Command parameter contains the unknown sentence. This events is fired only if you set AllNMEA to true in the Commands property and there is an unknown sentence received. In this case you can write your own processing code inside this event handler.
OnWayPointReceive: TWayPointReceiveEvent = procedure(Sender: TObject; const WayPoint: TWayPoint) - fires when a waypoint is received. WayPoint parameter contains the received waypoint.

ZylNMEAEncoder

ZylNMEAEncoder - Description Encodes position to NMEA format.

ZylNMEAEncoder - Properties:

Position - position to be encoded.
WayPoints - waypoints to be encoded.
NMEAPrefix - NMEA sentence prefix.
NMEATerminator - NMEA sentence terminator.

ZylNMEAEncoder - Methods:

Constructors/Destructors:

constructor Create(AOwner: TComponent) - constructor
destructor Destroy - destructor

Encoding methods:

function GetNMEACheckSum(const strNMEA: AnsiString): AnsiString - get the checksum of the NMEAsentence.
function AddCheckSumToNMEA(const strNMEA: AnsiString): AnsiString - add checksum to the NMEA sentence.
function GetGPGSA(): AnsiString - encodes GPGSA sentence.
function GetGPGGA(): AnsiString - encodes GPGGA sentence.
function GetGPRMC(): AnsiString - encodes GPRMC sentence.
function GetGPWPL(): AnsiString - encodes GPWPL sentence.
function GetNMEA(): AnsiString - encodes in NMEA using all supported sentences.

http://www.zylsoft.com
info@zylsoft.com

Zyl soft

TGPSPosition

TGPSPosition - Description:

TGPSPosition is a class which contains information about the current position.

TGPSPosition - Properties:

Latitude_Degree - degree component of latitude (0-90)
Latitude_Minute - minute component of latitude (0-59)
Latitude_Second - second component of latitude (0-59.(9))
Latitude_Direction - direction component of latitude (north or south)
Longitude_Degree - degree component of longitude (0-180)
Longitude_Minute - minute component of longitude (0-59)
Longitude_Second - second component of longitude (0-59.(9))
Longitude_Direction - direction component of longitude (east or west)
Altitude - altitude in meters
Heading - true heading in degrees
Course - true course in degrees
Speed_Knots - speed in knots
Speed_KMH - speed in km/h
Magnetic_Variation - magnetic variation in degrees
Magnetic_Variation_Direction - direction of magnetic variation (east or west)
Magnetic_Deviation - magnetic deviation in degrees
Magnetic_Deviation_Direction - direction of magnetic deviation (east or west)
Magnetic_Heading - magnetic heading in degrees
Magnetic_Course - magnetic course in degrees
Day - day component of date
Month - month component of date
Year - year component of date
Hour - hour component of time
Minute - minute component of time
Second - second component of time

Millisecond - millisecond component of time
Satellite_Count - count of the satellites used to determine the fix position
PDOP - position dilution of precision
HDOP - horizontal dilution of precision
VDOP - vertical dilution of precision
Geo_Height - difference between WGS-84 reference ellipsoid surface and the mean-sea-level altitude
Fix - info about fix (0 = Invalid, 1 = Valid SPS, 2 = Valid DGPS, 3 = Valid PPS)
Mode1 - 1 = Auto 2D/3D, 2 = Forced 2D/3D
Mode2 - 1 = no fix, 2 = 2D fix, 3 = 3D fix
Satellites - list of satellites in view

TGPSPosition - Methods:

constructor Create - constructor
procedure Reset - resets position paramters to default values
destructor Destroy - destructor
function GetAltitude: Extended - returns altitude in meters
function GetLatitudeAsDecimalDegrees: Extended - returns latitude in decimal degrees
function GetLatitudeAsDM: Extended - returns latitude in DM
function GetLatitudeAsRadians: Extended - returns latitude in radians
function GetLatitudeDegree: Integer - returns latitude degree
function GetLatitudeMinute: Integer - returns latitude minute
function GetLatitudeSecond: Extended - returns latitude second
function GetLatitudeDirection: TCardinalPoint - returns latitude direction
function GetLocalDateTime: TDateTime - returns local datetime
function GetLongitudeAsDecimalDegrees: Extended - returns longitude in decimal degrees. It's positive for east and negative for west.
function GetLongitudeAsDM: Extended - returns longitude in DM
function GetLongitudeAsRadians: Extended - returns longitude in radians
function GetLongitudeDegree: Integer - returns longitude degree
function GetLongitudeMinute: Integer - returns longitude minute
function GetLongitudeSecond: Extended - returns longitude second
function GetLongitudeDirection: TCardinalPoint - returns longitude

direction
function GetUTCDateTime: TDateTime - returns UTC datetime

function DecimalDegreestoRadians(Degree: Extended): Extended - converts decimal degrees to radians
function DMSToDecimalDegrees(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to decimal degrees
procedure DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended) - converts DMS (DegreeMinuteSecond) to DM (DegreeMinute)
function DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended): Extended- converts DMS (DegreeMinuteSecond) to DM (DegreeMinute - concatenated format: DDMM.mmmm)
function DMSToRadians(Degree, Minute: Integer; Second: Extended; Direction: TCardinalPoint): Extended - converts DMS (DegreeMinuteSecond) to radians
procedure DMToDMS(dmDegree: Integer; dmMinute: Extended; var Degree, Minute: Integer; var Second: Extended) - converts DM (DegreeMinute) to DMS (DegreeMinuteSecond)
procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for latitude values
procedure LatitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for latitude values
procedure LongitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts decimal degrees to DMS for longitude values
procedure LongitudeRadiansToDMS(Radian: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint) - converts radians to DMS for longitude values
function RadiansToDecimalDegrees(Radian: Extended): Extended - converts radians to decimal degrees

info@zylsoft.com

TSatellite

TSatellite - Description:

TSatellite is a class which contains information about the satellites in view of the GPS system.

TSatellite - Properties:

PseudoRandomCode - ID of the satellite (1-32)
Elevation - Elevation of the satellite in degrees (0-90)
Azimuth - Azimuth of the satellite in degrees (0-359)
SignalToNoiseRatio - Signal to noise ration in dBHZ (0-99)
SatelliteType - the type of the satellite (GP, GL, BD, GA, QZ, etc);

TSatellites

TSatellites - Description:

TSatellites is a container class of TSatellite objects.

TSatellites - Properties:

Count - count of satellites
Items[Idx: Integer]: TSatellite - saltellite items list

TSatellite - Methods:

constructor Create - constructor
destructor Destroy - destructor
procedure Add(const Item: TSatellite) - adds a new TSatelllite item at the end of the list
procedure Insert(Idx: Integer; const Item: TSatellite) - inserts a new TSatelllite item at the position specified by Idx parameter
procedure Delete(Idx: Integer) - deletes the satellite from position Idx
procedure DeleteById(const SatelliteId: Integer ; SatelliteType: String) - deletes the satellite with PseudoRandomCode specified by the SatelliteId parameter and SatelliteType (GP, GL, BD, etc);
procedure Clear - deletes all the satellites from the list
procedure ClearByType(SatelliteType: String) - deletes all the satellites

of a certain type (GP, GL, BD, etc);
function IndexOf(SatelliteId: Integer; SatelliteType: String): Integer - returns the list position of the satellite with PseudoRandomCode specified by the SatelliteId parameter and SatelliteType (GP, GL, BD, etc);
function GetById(SatelliteId: Integer; SatelliteType: String): TSatellite - returns the satellite with PseudoRandomCode specified by the SatelliteId parameter and SatelliteType (GP, GL, BD, etc);

TNavigationInfo

TNavigationInfo - Description:

TNavigationInfo is a class which contains information about navigational data, when a destintation waypoint is defined.

TNavigationInfo - Properties:

Origin - name of the origin waypoint
Destination - name of the destination waypoint
RangeToDestination - range to destintation in nautical miles
TrueBearingToDestination - true bearing to destination in degrees
MagneticBearingToDestination - magnetic bearing to destination in degrees
VelocityTowardsDestination - velocity towards destination in knots
CrossTrackError - cross-track error in nautical miles (9.9 max.)
DirectionToSteer - steer to correct the cross-track error (left or right)

TNavigationInfo - Methods:

constructor Create - constructor
procedure Reset - resets navigation paramters to default values
destructor Destroy - destructor

http://www.zylsoft.com
info@zylsoft.com

TReceiverStatus

TReceiverStatus - Description:

TReceiverStatus is a class which contains information about the status of the GPS receiver.

TReceiverStatus - Properties:

SignalStrength - signal strength in dB
SNR - signal to noise ration in dB
Frequency - beacon frequency (kHz)
DataRate - beacon data rate (BPS)

TReceiverStatus - Methods:

constructor Create - constructor
procedure Reset - resets receiver paramters to default values
destructor Destroy - destructor

http://www.zylsoft.com
info@zylsoft.com

TWayPoint

TWayPoint - Description:

TWayPoint is a class which contains information about one waypoint.

TWayPoint - Properties:

Name - name of the waypoint
Latitude_Degree - degree component of latitude (0-90)
Latitude_Minute - minute component of latitude (0-59)
Latitude_Second - second component of latitude (0-59.(9))
Latitude_Direction - direction component of latitude (north or south)
Longitude_Degree - degree component of longitude (0-180)
Longitude_Minute - minute component of longitude (0-59)
Longitude_Second - second component of longitude (0-59.(9))
Longitude_Direction - direction component of longitude (east or west)

TWayPoint - Methods:

constructor Create - constructor.
destructor Destroy - destructor.

TWayPoints

TWayPoints - Description:

TWayPoints is a container class of TWayPoint objects.

TWayPoints - Properties:

Count - count of waypoints
Items[index: Integer]: TWayPoint - waypoint items list.

TWayPoints - Methods:

constructor Create - constructor
destructor Destroy - destructor
procedure Add(const item: TWayPoint) - adds a new TWayPoint item at the end of the list
procedure Insert(index: Integer; const item: TWayPoint) - inserts a new

TWayPoint item at the position specified by index parameter
procedure Delete(index: Integer) - deletes the waypoint from position index
procedure Delete(const WayPointName: AnsiString) - deletes the waypoint with name specified by the WayPointName parameter
procedure Clear - deletes all the waypoints from the list
function IndexOf(const WayPointName: AnsiString): Integer - returns the list position of the waypoint with name specified by the WayPointName parameter
function GetByName(const WayPointName: AnsiString): TWayPoint - returns the waypoint with name specified by the WayPointName parameter

TRoute

TRoute - Description:

TRoute is a class which contains information about a route.

TRoute - Properties:

Name - name of the route
WayPoints - TStringList which contains the name of the waypoints from the route
Destination - name of destination waypoint

TRoute - Methods:

constructor Create - constructor
destructor Destroy - destructor
function WayPointListToStr: AnsiString - returns a comma separated list of waypoits from the route

http://www.zylsoft.com
info@zylsoft.com

TTrack

TTrack - Description:

TTrack is a class which contains information about the last registered track. The track contains the received positions.

TTrack - Properties:

Count - count of track positions.
Items - collection of the GPS positions.

TTrack - Methods:

constructor Create - constructor.
destructor Destroy - destructor.
procedure Add(const Item: TGPSPosition) - adds a new position at the end of the track list.
procedure Insert(index: Integer; const Item: TGPSPosition) - insterts a new position at the specified index.
procedure Delete(index: Integer) - delets a position from the specified index.
procedure Clear - clears all the positions of the track.

http://www.zylsoft.com
info@zylsoft.com