

## ZylNMEAGenerator 1.64



ZylNMEAGenerator is a Delphi / C++ Builder component that transforms position data in NMEA format. NMEA is a standard protocol, used by GPS receivers to transmit data.

You can generate complex NMEA sentences from simple data inputs. The sentences are recognized by any NMEA compatible mapping software. In this way with this component you can simulate either a gps receiver. You just write the generated NMEA sentences to a virtual serial port or an internet socket, connected to mapping software.

You can use this component also as a GPS simulator using a null-modem cable or a virtual serial port software.

### **Supported Operating Systems:** Windows

95/98/Me/NT/2000/XP/Server2003/Vista/Server2008/7/8/Server2012/10

**Available for:** Delphi 10.1 (Win32 & Win64), Delphi 10 (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 2005, Delphi 7, Delphi 6, Delphi 5, Delphi 4, 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, C++Builder 5, 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++

### **Installation:**

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 ZylNMEAGenPack package in the list and click the Remove button
- Open Tools - Environment Options - Library and remove the library path pointing to ZylNMEAGen 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 ZylNMEAGenerator
- Delete or rename the top folder where ZylNMEAGenerator 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 ZylNMEAGenerator. (<compiler> is either "Delphi" or "C++Builder", <version> is the IDE version you have installed)

-Unzip the zip file and open the ZylNMEAGenPack.dpk file in Delphi (ZylNMEAGenPack.bpk 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 Delphi menu) or Tools/Options/C++ Options/Paths and Directories/Library Path & Include Path (in newer C++Builder menu) the path of the installation (where the ZylNMEAGenerator.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".

-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 following 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.

#### **Properties:**

**Latitude\_Degree** - latitude degree

**Latitude\_Minute** - latitude minute

**Latitude\_Second** - latitude second

**Latitude\_Direction** - latitude direction (North/South)

**Longitude\_Degree** - longitude degree

**Longitude\_Minute** - longitude minute

**Longitude\_Second** - longitude second

**Longitude\_Direction** - longitude direction (East/West)

**Altitude** - altitude in meters

**Heading** - heading (course) in degrees

**Speed** - speed in knots

**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

**Magnetic\_Variation** - magnetic variation in degrees

**Magnetic\_Variation\_Direction** - direction of magnetic variation (East/West)

**Magnetic\_Heading** - magnetic heading in degrees

**UTCDateTime** - UTC date and time

**Commands** - enable/disable NMEA sentences

**UseSystemDateTime** - when this property is true, then the UTCDateTime property is calculated from the system datetime

**MessageAlert** - enable/disable message alerts

**Satellites:** TSatellites - satellite list

**WayPoints:** TWayPoints - waypoint list

**Routes:** TRoutes - route list

### Methods:

**constructor Create(AOwner: TComponent)** - constructor

**destructor Destroy()** - destructor

**procedure AddSecondsToLatitude(seconds: double)** - adds a number of seconds to the current latitude

**procedure AddSecondsToLongitude(seconds: double)** - adds a number of seconds to the current longitude

**procedure SetSatellite(p\_PseudoRandomCode: Word; p\_Active: Boolean = True; p\_Elevation: Word = 0; p\_Azimuth: Word = 0; p\_SignalToNoiseRatio: Word = 0)** - sets the parameters of the satellite identified by the parameter p\_PseudoRandomCode

**function GetSatellites: TSatellites** - returns the array of satellites

**function GetSatellitesList: TList** - returns the list of satellites

**function GetActiveSatellitesCount: Word** - returns the count of active satellites

**procedure SaveSatellitesToFile(fileName: String)** - saves the parameters of satellites to file

**procedure LoadSatellitesFromFile(fileName: String)** - loads the parameters of satellites from file

**procedure ResetSatellites()** - resets the parameters of satellites to the default values

**procedure DrawSatellites(Canvas: TCanvas; Radius: Integer; BrushColor, PenColor: TColor)** - draws the satellites to the canvas

**function GetNMEA: AnsiString** - returns the whole NMEA format of the position data

**function GetGPZDA: AnsiString** - returns NMEA command GPZDA

**function GetGPGGA: AnsiString** - returns NMEA command GPGGA

**function GetGPGLL: AnsiString** - returns NMEA command GPGLL

**function GetGPGSA: AnsiString** - returns NMEA command GPGSA

**function GetGPRMC: AnsiString** - returns NMEA command GPRMC

**function GetGPVTG: AnsiString** - returns NMEA command GPVTG

**function GetGPGSV: AnsiString** - returns NMEA command GPGSV

**function GetLatitudeAsDecimalDegrees: Extended** - returns latitude in decimal degrees

**function GetLongitudeAsDecimalDegrees: Extended** - returns longitude in decimal degrees

**function KmToKnots(pSpeed: Double): Double** - converts km/h in knots

**function KnotsToKmh(pSpeed: Double): Double** - converts knots in km/h

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

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

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

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

**function DMSToDecimalDegrees(Radian: Extended): Extended** - converts DMS to decimal degrees

**procedure LatitudeDecimalDegreesToDMS(DecDegree: Extended; var Degree, Minute: Integer; var Second: Extended; var Direction: TCardinalPoint)** - converts decimal degrees 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 DMSToDM(dmsDegree, dmsMinute: Integer; dmsSecond: Extended; var Degree: Integer; var Minute: Extended)** - converts DMS to DM

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

**function Connect(SerialPortName: AnsiString; BaudRateValue: Integer): Boolean** - connects to a serial port, with the specified baud rate value.

**function Connect(SerialPortName: AnsiString): Boolean** - connects to a serial port.

**procedure Disconnect** - disconnects from a serial port

**function GetSerialPort: AnsiString** - returns the serial port where the component is connected to

**function GetBaudRate: Integer** - returns the baud rate of the communication with the serial port where the component is connected to

**function Send(str: AnsiString): Cardinal** - sends data to a serial port and return the number of sent bytes

**procedure SendOnNewThread(str: AnsiString)** - sends data to a serial port using a separate thread

**function Transmit: Cardinal** - transmits the position data to a serial port and return the number of sent bytes

**procedure TransmitOnNewThread** - transmits GPS data to a serial port using a separate thread

### Types:

**TCardinalPoint** = (cpNorth, cpSouth, cpEast, cpWest)

**TNMEACommands** = set of (GPGLL, GPGGA, GPVTG, GPRMC, GPGSA, GPGSV, GPZDA, GPWPL, GPRTE, GPAAM)

**TSatellite** = class

property PseudoRandomCode: Word; //1-32

property Active: Boolean;

property Azimuth: Word; //0-360

property Elevation: Word; //0-90

property SignalToNoiseRatio: Word; //0-99

constructor Create(pseudoRandomCode: Word);

**end**

**TSatellites** = Array[1..32] of TSatellite

**TWayPoint** = class

property Name: AnsiString; //Name of the waypoint. Must be unique.

property Latitude\_Degree: Integer;

property Latitude\_Minute: Integer;

property Latitude\_Second: Extended;

property Latitude\_Direction: TCardinalPoint;

property Longitude\_Degree: Integer;

property Longitude\_Minute: Integer;

property Longitude\_Second: Extended;

property Longitude\_Direction: TCardinalPoint;

constructor Create(const WayPointName: AnsiString);

**end**

**TWayPoints** = class //Waypoint collection

property Count: Integer; //Count of waypoints

property Items[Index: Integer]: TWayPoint;

constructor Create();

destructor Dispose(DisposeItems: Boolean);

procedure Free;

procedure Add(const Item: TWayPoint);

procedure Insert(Idx: Integer; const Item: TWayPoint);

procedure Delete(Idx: Integer; DisposeItems: Boolean); overload;

procedure Delete(Idx: Integer); overload;

procedure Delete(const WayPointName: AnsiString); overload;

procedure Delete(const WayPointName: AnsiString; DisposeItems: Boolean); overload;

procedure Clear(DisposeItems: Boolean); overload;

procedure Clear; overload;

procedure Swap(const SourceName, DestinationName: AnsiString);

function IndexOf(const WayPointName: AnsiString): Integer;

function GetByName(const WayPointName: AnsiString): TWayPoint;

procedure SaveToFile(const FileName: String);

procedure LoadFromFile(const FileName: String);

**end**

**TRoute = class**

```

property Name: AnsiString; //Name of the route
property WayPoints: TStringList; //List of waypoint names
property Destination: AnsiString read GetDestination; //Destination waypoint's name
constructor Create(const RouteName: AnsiString);
destructor Destroy; override;
procedure SaveToFile(const FileName: String);
procedure LoadFromFile(const FileName: String);
function WayPointListToStr(): AnsiString; //returns comma separated string of waypoint names
end

```

**TRoutes = class** //Route collection

```

property Count: Integer; //Count of routes
property Items[Index: Integer]: TRoute;
property ActiveRoute: TRoute; //Active route
constructor Create();
destructor Destroy; override;
procedure Add(const Item: TRoute);
procedure Insert(Idx: Integer; const Item: TRoute);
procedure Delete(Idx: Integer); overload;
procedure Delete(const RouteName: AnsiString); overload;
procedure Clear;
function IndexOf(const RouteName: AnsiString): Integer;
procedure Swap(const SourceName, DestinationName: AnsiString);
function GetByName(const RouteName: AnsiString): TRoute
end

```

```
//custom exception class
```

```
EZylNMEAException = class(Exception);
```

[Buy Now!](#)

**Copyright by Zyl Soft 2003 - 2016**

<http://www.zylsoft.com>

[info@zylsoft.com](mailto:info@zylsoft.com)

