

Franson GpsGate <http://franson.com/gpsgate>

© 2004-2005 Franson Technology AB, All rights reserved

## Franson GpsGate API Reference Manual

Using GpsGate API you can start GpsGate, send data to and receive data from GpsGate. For OEM licensees this API is also used to activate GpsGate.

Applications can (of course) access or write data to GpsGate using any of the standard options in the GpsGate Settings menu, like serial ports, TCP/IP and UDP. The main advantage of using this API is that you can make sure GpsGate is started and you can access GPS data without knowing anything about how GpsGate (or the GPS) is configured.

To start, send and receive data read more about the [Simple](#) class.

For OEM license activation read more about the [L](#) class.

Configure GpsGate from you installation program, or application, see [Registry](#).

**Samples.** You find samples in GpsGate SDK which can be found on the beta [download page](#).

**Error handling.** The ActiveX component generates standard COM/OLE exceptions. In Visual Basic you typically handle those errors with "On Error Goto ..." and similar statements. The .NET component generates standard .NET exceptions, typically handled with try/catch statments. The C++ library do not generate exceptions, you will need to call GetLastError() to check if an error has ocured. Note that this is not the win32 version of GetLastError() rather a method of the used object.

**All variables** in the ActiveX components are **Variants**. This is to support all kinds of COM enabled script languages. The type information (e.g. String, Integer, etc.) that are specified in the reference manual are the Variant type and nothing else. For Visual Basic and script language developer this is of no major concern, everything is handled automatically, but it can be good to know. In the .NET and C++ static libraries no variants are used.

### Name of component

.NET (Desktop + Compact)	ActiveX Windows	ActiveX Pocket PC
GateApiNET	GateApiXP	GateApiCE

**Technical support** can be found in the user forum. We will constantly monitor and answer questions in the forum. The forum also includes frequently asked question (FAQ).

[FAQ for GpsGate](#)

[Browse the Technical support forum!](#)

[Search the Technical support forum!](#)

<b>Class</b> <a href="#">Simple</a>	<b>Class</b> <a href="#">L</a>
<b>Methods</b> <a href="#">StartGpsGate</a> <a href="#">Open</a> <a href="#">Close</a> <a href="#">Write</a>  <a href="#">Read</a> <a href="#">GetLastError</a>	<b>Methods</b> <a href="#">Reg</a>  <a href="#">GetLastError</a>
<a href="#">ByteArrayToString</a> <a href="#">StringToByteArray</a>	
<b>Events</b> <a href="#">OnRead</a>	
<b>Properties</b> <a href="#">MultiThreading</a> <a href="#">Parent</a>	

## Simple GpsGate v1.10 and later

The Simple class is used to start GpsGate and to read data from and write data to GpsGate.

You can make sure GpsGate is started by using `Simple.StartGpsGate`. To open a data stream to/from GpsGate use `Simple.Open`.

## Simple.StartGpsGate GpsGate v1.10 and later

Starts GpsGate if it is not yet started. After GpsGate is confirmed to be up and running the method returns. If StartGpsGate fails to start an error is generated.

### Syntax

```
object.StartGpsGate()
```

## Simple.Open GpsGate v1.10 and later

Opens a data connection to GpsGate.

If "**\_output**" is specified as argument a connection is established to the data coming out from GpsGate. This is the same (NMEA) data you would get by connecting to one of GpsGate's virtual serial ports. The main advantage of using the GpsGate API instead of connecting to a virtual serial port is that your application do not

need to know anything about how GpsGate nor the GPS is configured to access GPS data. Any number of applications can connect to "\_output" at the same time.

If "\_input" is passed as argument a connection to GpsGate input is made. This means that GpsGate will take the data sent by your application as input, and send that data to its output ports. This is very usefull if you want to control the GPS input to other applications or if you want to use the virtual serial ports for some custom usage. "\_input" can only be opened exclusively and GpsGate must be configured to use **Gate Direct** as input. You find this option under GpsGate Settings in the tray menu. This must be set manually by the end user.

A call to `Simple.StartGpsGate` is recommended before `Open` is called to make sure GpsGate is started. If the connection fails an error is generated.

After having successfully opened a connection data can be read and written using `Simple.Write`, `Simple.OnRead` and `Simple.Read`.

To close the connection use `Simple.Close`.

## Syntax

`object.Open(Name)`

Part	Type ActiveX	Type .NET	Type C++	Description
<i>Name</i>	Variant (String)	string	char*	Name of connection, "_output" or "_input".

## Simple.Close

GpsGate **v1.10** and later

Closes an opened connection.

## Syntax

`object.Close()`

## Simple.Write

GpsGate **v1.10** and later

Writes data to an opened connection.

`Write` can handle both string and binary data. If you write binary data a translation from an array to string first needs to be made. Use one of the following methods:

## ActiveX (Visual Basic)

```
Dim str As String
Dim arr(2) As Byte      ' Could be any length

arr(0) = 0             ' Binary data to be written to GpsGate
arr(1) = 34
arr(2) = 2

str = StrConv(arr, vbUnicode)  ' Convert to string
simple.Write(str)              ' Write data
```

**.NET****(VB.NET)**

```
Dim binary_data As Byte() = {&H81, &H82}

' Write binary data
objSimple.Write(GateApiNET.Simple.ByteArrayToString(byte_array))
```

For

**C++ static library**

there is no difference between writing binary and string data.

Write

will generate an error if a connection is not opened, GpsGate has been closed, or if the connection is opened as "\_input" and the user has deselected "Gate Direct".

**Syntax**

(ActiveX and .NET)

*object*

.Write(Data)

Part	Type ActiveX	Type .NET	Description
<i>Data</i>	Variant (String)	string	Data to be written to GpsGate connection.

**Syntax** (C++ static library)

*object*.Write(Data, Len)

Part	C++	Description

<i>Data</i>	char*	Pointer to buffer of data to be written to GpsGate connection. Do not need to be NULL terminated. And may contain pure binary data.
<i>Len</i>	int	Length of buffer. Should not include (an optional) NULL terminating character

## Simple.Read

GpsGate v1.10 and later

**NOTE!** This method is only implemented in the C++ static library. ActiveX and .NET implementations should use `Simple.OnRead`.

Read receives data from an opened connection.

**Timeout.** If Timeout is set to -1 Read blocks until any data is received or until an error occurs. If Timeout is set to 0 Read returns instantly with an empty buffer if no data was to receive. If Timeout is set to a positive value, Read waits Timeout miliseconds or until there is any data to receive until it returns. Note that Read does not wait until the buffer is filled up, it always returns if there is data to receive.

**Error.** If the connetion is closed for some reason false is returned. The connection can be closed by one of the folloing reasons:

1. GpsGate was closed.
2. If connection was opened as "\_input" and the user deselects "Gate Direct" as input under GpsGate Settings.
3. The application called `Simple.Close`. (Which is a good way to cancel a blocking call to Read)

After an error the connection cannot be used anymore. A new connection must be opened. In case 1 and 2 `Simple.Close` must be called. You can call `Simple.GetLastError` to get more error information.

### Syntax

status = *object*.Read(Buffer, BufferLen, ReceivedLen, Timeout)

Part	Type	Description
<i>status</i>	bool	Returns flase on error in all other cases (including timeout) true is returned.
<i>Buffer</i>	char*	Pointer to buffer to receive data
<i>BufferLen</i>	int	Size of buffer in bytes.
<i>ReceivedLen</i>	int*	Pointer to an int that contains number of bytes written to buffer on return.
<i>Timeout</i>	int	See description above.

## Simple.GetLastError

GpsGate v1.10 and later

**NOTE!** This method is only implemented in the C++ static library. ActiveX and .NET uses exceptions to return errors.

GetLastError (not to be confused with the win32 version with the same name) should be called after each function call to check if an error ocured.

### Syntax

code = *object*.GetLastError(Buffer, BufferLen)

Part	Type	Description
<i>code</i>	bool	Returns error code. Zero is returned if no error has occurred.
<i>Buffer</i>	char*	Pointer to buffer to receive error message. The string is NULL terminated
<i>BufferLen</i>	int	Size of buffer in bytes.

## Simple.OnRead

GpsGate v1.10 and later

Receives data from an opened connection.

**Note!** This event is only supported for ActiveX and .NET. For C++ static library use `Simple.Read`

`OnRead` can handle both string and binary data. If you want to translate the received data to binary data use one of the following methods:

### ActiveX (Visual Basic)

```
Sub objSimple_OnRead(str As String)
Dim arr() As Byte          ' Could be any length

arr = StrConv(str, vbFromUnicode)      ' Convert to byte array
```

### .NET

#### (VB.NET)

```
Sub objSimple_OnRead(string_data As String)
Dim binary_data() As Byte

' Received binary data
binary_data = GateApiNET.Simple.StringToByteArray(string_data)
```

The argument to

`OnRead`

will be NULL if the connection has been closed. This can be caused by: 1. GpsGate was closed.

2. If connection was opened as "\_input" and the user deselects "Gate Direct" as input under GpsGate Settings.

3. The application called

`Simple.Close`

.

After an error the connection cannot be used anymore. A new connection must be opened. In case 1 and 2

`Simple.Close`  
must be called.

For  
**.NET applications**  
it is  
**important**  
to take a look at  
`Simple.Parent`  
.

**Syntax**  
(ActiveX and .NET)

`objSimple_OnRead(Data)`

Part	Type ActiveX	Type .NET	Description
<i>Data</i>	Variant (String)	string	Data received from GpsGate connection.

## **Simple.ByteArrayToString**

GpsGate **v1.10** and later

**NOTE!** This method is only implemented in the .NET component. See `Simple.Write` for more information on writing binary data.

If you want to write anything else than ascii strings (that is ascii value 0-127) you first need to create a byte array and then convert the array to a string using this **static** function. The string can then be used in `Simple.Write`

VB.NET sample:

```
Dim binary_data As Byte() = {&H81, &H82}

' Write binary data
objSimple.Write(GateApiNET.Simple.ByteArrayToString(byte_array))
```

Note that it will not work to build your own "binary string" using `chr()`. You must use this function!

See also  
`Simple.StringToByteArray`  
.

## Syntax

```
str
= GateApiNET.Simple.ByteArrayToString (
ByteArray
)
```

Part	Type	Description
<i>str</i>	string	Byte array converted to a string.
<i>ByteArray</i>	byte[]	Byte array to be converted.

## Simple.StringToByteArray

GpsGate **v1.10** and later

**NOTE!** This method is only implemented in the .NET component. See `Simple.OnRead` for more information on reading binary data.

If you want to read anything else than ascii strings (that is ascii value 0-127) you need to convert the received string to a byte array using this **static** function. The string has been received by `Simple.OnRead`.

VB.NET sample:

```
Sub objSimple_OnRead(string_data As String)
Dim binary_data() As Byte

' Received binary data
binary_data = GateApiNET.Simple.StringToByteArray(string_data)
```

Note that it will not work to build your own byte array, e.g. using `String.ToCharArray()`. You must use this function.

See also

`Simple.ByteArrayToString`

## Syntax

```
ByteArray
= GateApiNET.Simple.StringToByteArray (
Str
)
```

Part	Type	Description
<i>ByteArray</i>	byte[]	Byte array converted from a string.
<i>Str</i>	string	String to be converted.

## Simple.MultiThreading

GpsGate **v1.10** and later

This property is only supported for ActiveX

If set to False (default), the event is called in the same thread as the form (GUI) is running in. This is what you want when using GpsGate API in a form based application.

If set to True, the event is called in a new thread. This is what you want if you are using GpsGate API in a non-GUI (form-less) application like a service. Or if you decide to wrap GpsGate API into your own custom classes.

**Note!** Visual Basic 6.0 do not support multi-threading, this means your application will crash if you set MultiThreading to True when using GpsGate API as part of your VB 6.0 application.

**Note!** Since the event is called in a new thread you must make sure your code is thread safe when setting this property to True.

For .NET the same functionality is achieved using `Simple.Parent`

### Syntax

*object*.MultiThreading = *Value*

*Value* = *object*.MultiThreading

Part	Type ActiveX	Type .NET	Description
<i>Value</i>	Variant (Boolean)	bool	.

## Simple.Parent

GpsGate **v1.10** and later

This property is only supported for .NET

If this property is NULL (default) `Simple.OnRead` is called in a separate thread, this means your application becomes multi threaded. This is what you want if you are using GpsGate API in a non-GUI (form-less) application like a service. Or if you decide to wrap GpsGate API into your own custom classes.

If this property is set to your application form (System.Windows.Forms.Control) then OnRead is called in the GUI main thread, this means your application remains single threaded. This is what you want when using GpsGate API in a form based application.

### Syntax

*object*.Parent = *Value*

Part	Type .NET	Description
------	-----------	-------------

Value	System.Windows.Forms.Control	Your application form.
-------	------------------------------	------------------------

All settings for GpsGate is stored in the registry under HKEY\_CURRENT\_USER\Software\Franson\GpsGate. If you need to programatically control GpsGate settings, you can do that by changing the registry entries described below. GpsGate must be closed when changes in the registry are made. You can check if GpsGate is closed using `Simple.Open`.

Name	Description
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\Input\</b>	
Input, REG_DWORD	Which source is used as input to GpsGate. 0 - No source 1 - Not used. 2 - COM port 3 - NMEA Logger 4 - GPS simulator 5 - Not used 6 - UDP 7 - TomTom API 8 - TCP/IP 9 - Virtual COM port 10 - GpsGate API "_input" 11 - Garmin USB
Baudrate REG_DWORD	Baudrate used if input is set to COM port (1). Default 4800
Port REG_DWORD	COM Port number used if COM port (1) is set as input. Default 1
RetryConnect REG_DWORD	Algorithm used by GpsGate to reconnect to any source. 1 - Reconnect on error. 2 - Do not reconnect 3 - Reconnect on timeout. Default.
TcpHostname REG_SZ	Name or IP address of TCP/IP server. Used when input source set to TCP/IP (8)
TcpPort REG_DWORD	Port on TCP/IP server. Used when input source set to TCP/IP (8)
UdpPort REG_DWORD	UDP port. Used when input set to UDP (6)
VirtualPort REG_DWORD	Virtual com port used for inout. Used when input set to Virtual Port (9)
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\Output\</b>	

COMx REG_SZ	Defines all virtual com ports used as output, to which GPS applications can connect. The name of the variables are the names of the ports to be created. The values should always be "virtual" E.g. "COM4" REG_SZ "virtual" "COM5" REG_SZ "virtual"
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\PhysicalOutput\</b>	
COMx REG_SZ	Defines all physical com ports used as output. The name of the variables are the names of the ports. The values should always be "physical" E.g. "COM1" REG_SZ "physical"
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\TcpOutput\</b>	
x REG_SZ	Defines all TCP/IP ports used as output. The name of the variables are the names of the ports. The values should always be "Tcp" E.g. "20175" REG_SZ "Tcp"
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\Intervals\</b>	
GGA REG_SZ	Intervals for GGA sentences in seconds. Used by the simulator. Default 1.
GLL REG_SZ	Intervals for GLL sentences in seconds. Used by the simulator. Default 1.
RMC REG_SZ	Intervals for RMC sentences in seconds. Used by the simulator. Default 1.
VTG REG_SZ	Intervals for VTG sentences in seconds. Used by the simulator. Default 1.
GSA REG_SZ	Intervals for GSA sentences in seconds. Used by the simulator. Default 5.
GSV REG_SZ	Intervals for GSV sentences in seconds. Used by the simulator. Default 5.
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\Log\</b>	
PlayFilePath REG_SZ	Path of NMEA log file to play.
RecordFilePath REG_SZ	Path of NMEA log file to record to.
RealTime REG_DWORD	0 - Leave time stamps in NMEA sentences unchanged. Default 1 - Change time stamps in NMEA sentences to current (UTC) time during play.
RecordOnStart REG_DWORD	0 - Do not start recording when GpsGate starts. Default. 1 - Start recording log when GpsGate starts
Repeat REG_DWORD	0 - Do not repeat log file when it has reached its end. Default. 1 - Repeat log file from beginning when it has reached its end.
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\Sim\</b>	

SimFilePath REG_SZ	Path of simulator file.
SpeedUnit REG_DWORD	Which unit is displayed for speed in the simulator dialog 0 - Knots 1 - Meters per second 2 - Kilometers per hour 3 - Miles per hour
<b>HKEY_CURRENT_USER\Software\Franson\GpsGate\</b>	
OutputGroup, REG_DWORD	Which output group is displayed in the settings dialog 1 - Virtual ports 2 - Physical ports 3 - TCP/IP ports

© 2002-2005 Franson Technology AB, All rights reserved (franson.com)