March 9, 2016

GpsTrax

LOCATION BASED TRACKING AND MANAGEMENT

MATT GIESELMAN

GIESELMAN SOFTWARE | 1254 Holmgrove Drive, San Marcos, CA 92078 | gpstrax@gieselman.com

Table of Contents

Server System Requirements

- Microsoft Windows Server 2008 R2 or 2012 (x86 or x64, x64 recommended)
- Microsoft SQL Server 2008/2012/2014 (Only required if using MSSQLServerMessageHandler or EntityFrameworkMessageHandler)
- Microsoft .Net 4.5.1 Runtime

Developer System Requirements

- Microsoft Visual Studio 2013 or newer
- Windows 7 or newer

GpsTrax Architecture

GpsTrax consists of several components used to track, store and display data from GPS tracking units. The core software, Gieselman.Trax.dll, wrapped by a Windows Service is designed to abstract the hardware specific communication from higher level storage and display components. This abstraction allows GpsTrax to work with multiple hardware devices while providing a single interface for communication and data storage. GpsTrax allows OEMs to concentrate on writing their application logic from the start instead of working on supporting various hardware interfaces.

Additional flexibility is provided through the use of the provided plugins or the authoring your own to integrate GpsTrax into your existing environment and reducing time to market.

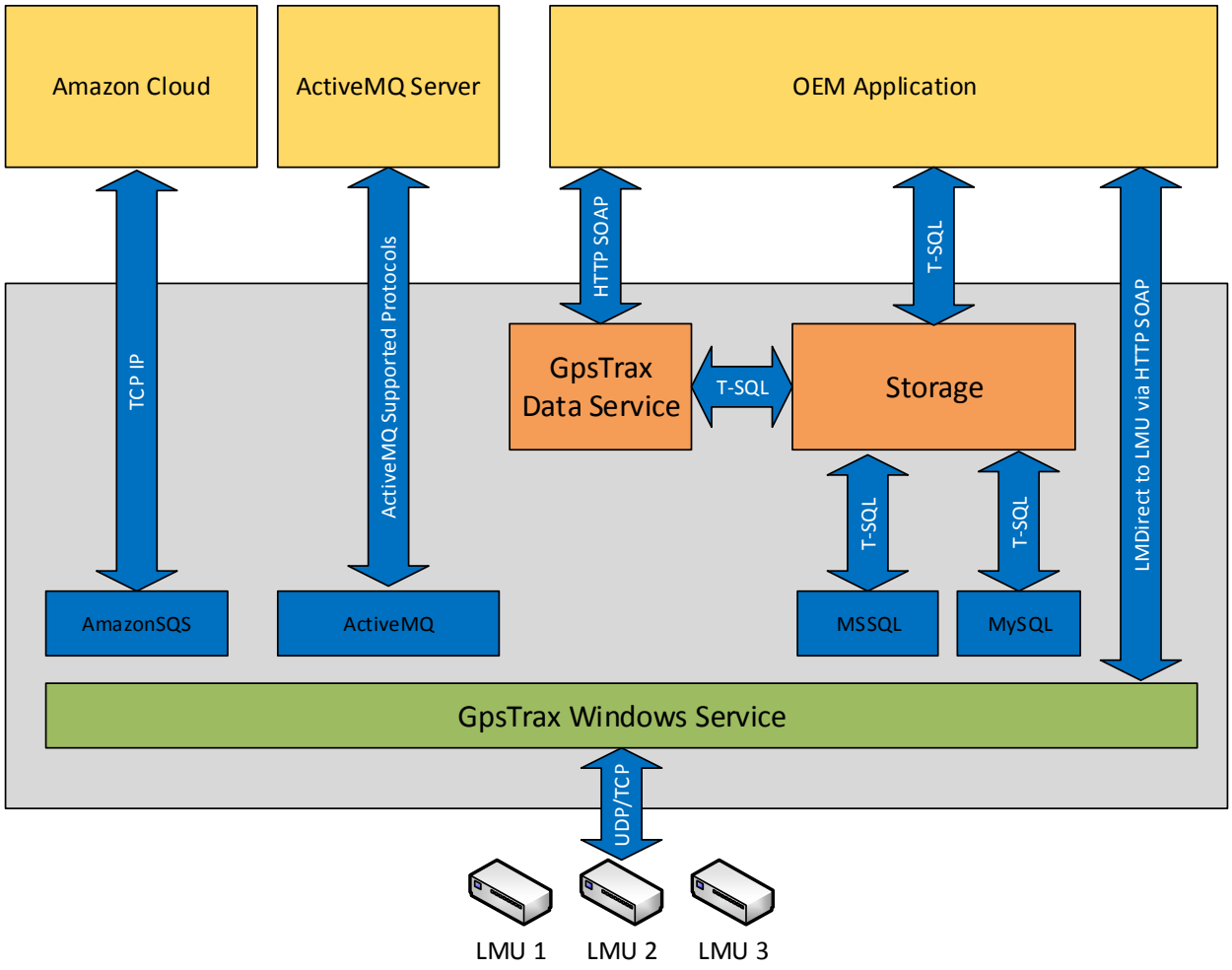

Figure 1 - GpsTrax System Architecture

GpsTrax includes the following components:

- GpsTrax Service – Windows service that wraps the Gieselman.Trax core library.
- GpsTrax Database – Default schema for data storage.

- GpsTraxMvc Web Application – Fully modern sample website using Javascript and HTML 5. Allows authenticated users to manage devices and visualize data using Google Maps.
- GpsTrax Wcf Service – Web service for pulling data from the MS SQL Schema using SOAP.
- Plugins for:
    - Active MQ
    - Amazon Kinesis
    - Amazon SQS
    - Azure Q
    - Entity Framework Code First Database
    - MSMQ
    - Microsoft SQL Server
    - MySQL
    - Push to web service
    - RabbitMQ
    - XML File

Each component is independent from the other allowing flexible deployment and software development. GpsTrax Service can be used as a standalone device communication module, add GpsTrax Web Service for 3rd party data access and GpsTrax Database for data storage and finally GpsTrax Web Application for data display and device management.

Gieselman.Trax.dll

The Gieselman.Trax.dll is the core GpsTrax component and includes all of the data types and logic to communicate with remote devices.

GpsTrax Service

GpsTrax Service acts as a Windows Service wrapper around Gieselman.Trax.dll by consuming device events and inserting them into the database schema. Additionally, the service exposes a SOAP interface to allow outbound messages to be sent to remote devices from external components.

GpsTrax Database

Database schema's for MS SQL and MySQL database engines.

GpsTraxWcf Web Service

A SOAP web service hosted in IIS exposing event records through a language neutral web invoke able interface.

An example SOAP request is shown below.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">http://tempuri.org/IEvents/GetEventReports
</Action>
  </s:Header>
  <s:Body>
    <GetEventReports xmlns="http://tempuri.org/">
      <mobileId>4431001121</mobileId>
      <startDate>2010-10-29T12:59:00</startDate>
      <endDate>2010-10-29T20:59:00</endDate>
    </GetEventReports>
  </s:Body>
</s:Envelope>
```

GpsTraxMvc Web Application

Example asp.net MVC 5 web application, that includes device management, reporting and Google maps integration.

Supported Message Types

GpsTrax supports all inbound message types including:

- Null (0)
- Ack (1)
- Event Report (2)
- ID Report (3)
- User (4)
- Application (5)
- Parameter Read (6)
- Unit Request (7)
- Locate Report (8)
- User Accumulate (9)
- Mini Event Report (10)
- Mini User (11)
- Mini Application (12)

GpsTrax also supports sending of the following outbound message types:

- Unit Request
- User Message
- Application Message
- Parameter Write

Installation and Management

GpsTraxService.exe

The GpsTrax Service can be installed by running GpsTraxInstaller.msi.

Next GpsTrax needs to be configured to use at least one plugin, copy the desired plugin from the GpsTrax installation folder to the Plugins folder also in the installation folder. For example to configure GpsTrax to use the MSSQL plugin you would copy MSSQLMessageHandler.dll and MSSQLMessageHandler.pdb to the Plugins folder.

After the plugin has been copied to the Plugins folder configure the appropriate section(s) in GpsTraxService.exe.config.

Once GpsTrax has been installed and the plugins configured the start the service by running the following command line:

```
net start gpstrax
```

The settings are shown below along with explanations of values.

```
<applicationSettings>

    <GpsTraxService.Properties.Settings>
      <!--The location that GpsTrax will look for plugins, by default it's in the same directory as
GpsTraxService.exe.-->
      <setting name="PluginsDirectory" serializeAs="String">
        <value>.\Plugins </value>
      </setting>
    </GpsTraxService.Properties.Settings>

    <Gieselman.Trax.Properties.Settings>
      <!--Increasing the receive buffer size increases the number of messages per second GpsTrax can
process.-->
      <setting name="ReceiveBufferSize" serializeAs="String">
        <value>512000</value>
      </setting>
      <!--The UDP port that GpsTrax will listen for LM Direct messages, normally port 20500.-->
      <setting name="LMUPort" serializeAs="String">
        <value>20500</value>
      </setting>
      <!--The TCP port that GpsTrax will listen for LM Direct messages over the Iridium network, normally
port 20501.-->
      <setting name="IridiumPort" serializeAs="String">
        <value>20501</value>
      </setting>
    </Gieselman.Trax.Properties.Settings>

    <!--AzureQ settings.-->
    <AzureQMessageHandler.Properties.Settings>
      <setting name="QueueName" serializeAs="String">
        <value>gpstrax</value>
      </setting>
      <setting name="StorageConnectionString" serializeAs="String">
        <value>DefaultEndpointsProtocol=https;AccountName=gpstrax;AccountKey=YOUR_KEY</value>
      </setting>
    </AzureQMessageHandler.Properties.Settings>

    <!--ActiveMQ settings.-->
    <ActiveMQMessageHandler.Properties.Settings>
```

```
  <setting name="QueueName" serializeAs="String">
    <value>GpsTrax</value>
  </setting>
  <setting name="BrokerUrl" serializeAs="String">
    <value>failover://(tcp://localhost:61616?jms.useAsyncSend=true)</value>
  </setting>
  <setting name="PersistentMode" serializeAs="String">
    <value>False</value>
  </setting>
</ActiveMQMessageHandler.Properties.Settings>

<!--AmazonSQS settings.-->
<AmazonSQSMessageHandler.Properties.Settings>
  <setting name="ServiceURL" serializeAs="String">
    <value></value>
  </setting>
  <setting name="AccessKey" serializeAs="String">
    <value></value>
  </setting>
  <setting name="SecretKey" serializeAs="String">
    <value></value>
  </setting>
  <setting name="QueueName" serializeAs="String">
    <value></value>
  </setting>
</AmazonSQSMessageHandler.Properties.Settings>

<!--MSMQ settings.-->
<MSMQMessageHandler.Properties.Settings>
  <setting name="QueueName" serializeAs="String">
    <value>.\private$\GpsTrax</value>
  </setting>
</MSMQMessageHandler.Properties.Settings>

<!--MSSQL settings.-->
<MSSQLMessageHandler.Properties.Settings>
  <setting name="IDReportRequestInterval" serializeAs="String">
    <value>2400</value>
  </setting>
</MSSQLMessageHandler.Properties.Settings>

<!--MySQL settings.-->
<MySQLMessageHandler.Properties.Settings>
  <setting name="ConnectionString" serializeAs="String">
    <value>Server=127.0.0.1;Database=GpsTrax;Uid=root;Pwd=YOUR_PASSWORD;</value>
  </setting>
</MySQLMessageHandler.Properties.Settings>

<!--RabbitMQ settings.-->
<RabbitMQMessageHandler.Properties.Settings>
  <setting name="HostName" serializeAs="String">
    <value>localhost</value>
  </setting>
  <setting name="InboundQueue" serializeAs="String">
    <value>Inbound</value>
  </setting>
  <setting name="Exchange" serializeAs="String">
    <value>GpsTrax</value>
  </setting>
  <setting name="UserName" serializeAs="String">
    <value />
  </setting>
  <setting name="Password" serializeAs="String">
    <value />
  </setting>
</RabbitMQMessageHandler.Properties.Settings>

<!--XML file settings.-->
```

```
    <XMLFileMessageHandler.Properties.Settings>
      <setting name="LogPath" serializeAs="String">
        <value>C:\logs\GpsTraxMessages.xml</value>
      </setting>
    </XMLFileMessageHandler.Properties.Settings>


  </applicationSettings>
```

Log

By default, GpsTraxService.exe logs to C:\Logs\GpsTrax.log, to change the log configuration edit the log4net section of GpsTraxService.exe.config

Database

GpsTrax Service supports a default schema compatible with SQL Server 2008 R2, 2012 or MySQL. To deploy the schema open the appropriate .sql file in SQL Server Management Studio and modify the `CREATE DATABASE` portion to use paths and setting appropriate to your environment.

```
CREATE DATABASE [GpsTrax]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'GpsTrax', FILENAME = N'C:\Dev\Data\GpsTrax\GpsTrax.mdf' , SIZE = 2211840KB , MAXSIZE =
UNLIMITED, FILEGROWTH = 2048KB )
 LOG ON
( NAME = N'GpsTrax_log', FILENAME = N'C:\Dev\Data\GpsTrax\GpsTrax_log.ldf' , SIZE = 1518976KB , MAXSIZE =
2048GB , FILEGROWTH = 10%)
```

Sample Code

Intialization of GpsTrax

GpsTrax can be started with a minimal amount of code by adding a reference to Gieselman.Trax.dll and making the following calls:

```
Tracker tracker = new Tracker();

tracker.MessageReceived += new MessageReceivedHandler(Tracker_MessageReceived);

tracker.Start();
```

As messages are received on the event handler will be called asynchronously, GpsTrax handles sending acknowledgements back to the LMU.

Each event handler is passed a class encapsulating all information returned by the LMU message. See CalAmp LMU documentation for details on the data returned by each message type.

Sending Outbound Messages

Sending most outbound messages requires a single method call:

```
if (!tracker.SendUserMessage(0, 0, Garmin, new IPEndPoint(IPAddress.Parse("10.15.1.21"), 20510),
10000))
    {
        Console.WriteLine("ACK not received.");
    }
```

For Parameter messages simply create the appropriate object type and then call `SendParameterWriteMessage`:

```
SpeedParameter speedParameter = new SpeedParameter(2458); // 55 mph = 2458 cm/s

 tracker.SendParameterWriteMessage(12345678, speedParameter, new
IPEndPoint(IPAddress.Parse("10.15.1.21"), 20510), 10000);
```

Documentation

GpsTrax includes a help file and extensive context sensitive help along with sample code showing how to exercise the GpsTrax API.

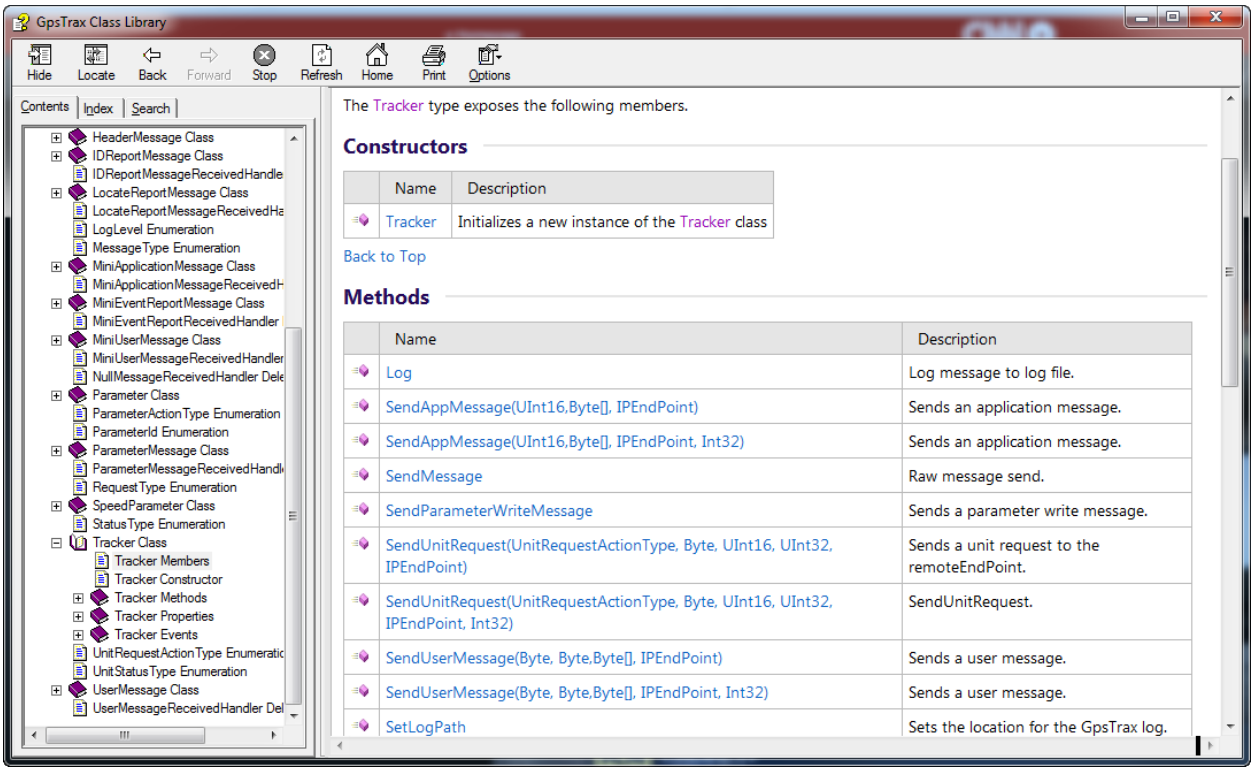

Figure 2 - GpsTrax Help File

Licensing

GpsTrax is available as either a binary or source license, the binary license does not include the source code to Gieselman.Trax.dll. The source license includes source code for all components.

All licenses include the following:

- Lifetime free software updates.
- 90 days of technical support.
- 5 hours of software development support.

Message Data Types

```
public class HeaderMessage
    {
        public DateTime ReceivedTimeStamp;
        public Request RequestType;
        public byte[] RawMessage;
        public LMDIRECT_HDR_STRUCT Header;
        public IPEndPoint Source;
        public MessageType MessageType
        public long MobileId
        public short SequenceNumber
        public Request RequestType
    }

public class EventReportMessage : HeaderMessage
    {
        public DateTime TimeStamp
        public DateTime TimeOfFix
        public decimal Latitude
        public decimal Longitude
        public int Speed
        public int Heading
        public decimal Altitude
        public byte Satellites
        public FixStatus FixStatus
        public short NetworkID
        public short RSSI
        public CommunicationState CommunicationState
        public decimal GPSHorizontalDilution
        public byte EventIndex
        public byte EventCode
        public uint[] Accumulators
        public byte Inputs
        public byte UnitStatus
    }

public class MiniEventReportMessage : HeaderMessage
    {
        DateTime TimeStamp;
        decimal Latitude;
        decimal Longitude;
        int Speed;
        int Heading;
        byte Satellites;
        FixStatus FixStatus;
        CommunicationState CommunicationState;
        byte EventIndex;
        byte EventCode;
        uint[] Accumulators;
        byte Inputs;
        byte UnitStatus;
   }

public class UserMessage : HeaderMessage
    {
        public DateTime TimeStamp;
```

```
        public DateTime TimeOfFix;
        public decimal Latitude;
        public decimal Longitude;
        public int Speed;
        public int Heading;
        public decimal Altitude;
        public byte Satellites;
        public FixStatus FixStatus;
        public short NetworkID;
        public short RSSI;
        public CommunicationState CommunicationState;
        public decimal GPSHorizontalDilution;
        public byte MessageRoute;
        public byte MessageId;
        public short Length;
        public byte[] Message;
        public byte Inputs;
        public byte UnitStatus;
        public uint[] Accumulators;
    }

public class ApplicationMessage : HeaderMessage
    {
        public DateTime TimeStamp;
        public DateTime TimeOfFix;
        public decimal Latitude;
        public decimal Longitude;
        public int Speed;
        public int Heading;
        public decimal Altitude;
        public byte Satellites;
        public FixStatus FixStatus;
        public short NetworkID;
        public short RSSI;
        public CommunicationState CommunicationState;
        public decimal GPSHorizontalDilution;
        public byte Inputs;
        public byte UnitStatus;
        public short ApplicationMessageType;
        public short Length;
        public byte[] Message;
    }

public class MiniUserMessage : HeaderMessage
    {
        public DateTime TimeStamp;
        public byte MessageRoute;
        public byte MessageId;
        public short Length;
        public byte[] Message;
    }

public class IDReportMessage : HeaderMessage
    {
        public byte ScriptVersion;
        public byte ConfigurationVersion;
        public string ApplicationVersion;
```

```
        public byte VehicleClass;
        public byte UnitStatus;
        public StatusType MemoryTest;
        public StatusType GPSAntenna;
        public StatusType GPSReceiverSelfTest;
        public StatusType GPSReceiverTracking;
        public StatusType ModemMINTestResults;
        public StatusType GPSException;
        public public byte ModemSelection;
        public byte ApplicationID;
        public byte MobileIDType;
        public uint QueryIdentifier;
        public string ESN;
        public string IMEI;
        public string IMSI;
        public string PhoneNumber;
        public string ICCID;
};
```