# Webservices am Beispiel GPS

**Çetin Öreten (13373)**

**Peter Fricker (12988)**

# Webservices am Beispiel GPS

Çetin Öreten (13373)
Peter Fricker (12988)

# Inhalt

1. Einleitung ..... 1
    1.1. Aufgabenstellung ..... 1
2. Analyse ..... 3
    2.1. AppWeb ..... 3
3. Design ..... 5
4. Implementierung ..... 7
    4.1. Portierung AppWeb ..... 7
    4.2. Implementierung GPS-Webservices ..... 7
    4.3. Webservice GetGPSData ..... 8
    4.4. Webservice GetGPSMap ..... 9
    4.5. GPS-Klasse ..... 10
    4.6. GPSParser-Klasse ..... 11
    4.7. Log-In ..... 13
A. ..... 17
Glossar ..... 19
Index ..... 21

# Abbildungsverzeichnis

1.1. Flea-System Spezifikation ..... 1
3.1. Aufruf GPS Webservice im Online Zustand (SSL) ..... 5
3.2. Aufruf GPS Webservice im Offline Zustand (SSL) ..... 6
4.1. GetGPSData.egi ..... 8
4.2. GetGPSMAP.egi ..... 9
4.3. GPS Class ..... 10
4.4. GPSParser Class ..... 11
4.5. Login ..... 14
4.6. Login processed ..... 15
4.7. Mapping KFZ <-> Hostname ..... 15

# Kapitel 1. Einleitung

Im Rahmen der Veranstaltung für Software-Praktikum, überlegen sich Studenten geeignete Themen aus, die sie bis Ende des Semesters realisieren. Lernziel soll die Fähigkeit zur selbstständigen Einarbeitung in aktuelle Technologie und deren Verwendung im Rahmen eines Projekts sein. Unser Thema war die Implementierung eines Webservice-Prototypen für das embedded Flea-System der Fa. CarMediaLab GmbH.

Primäre Aufgabe des Flea-Sytems besteht darin, Kommunikationswege zwischen dem Bussystem im Fahrzeug, dem Anwender und einem Backend zu etablieren. Die Nutzung dieser Wege erfordert außer Routingmechanismen auch Basisdienste zur sicheren Datenübertragung und Konnektivität. Mehrwert entsteht bei der Verwendung einer Telematikeinheit im Fahrzeug durch fahrzeugtypische Anwendungen, den sogenannten CarIntegrated Services. Das Flea-System ist so ausgelegt, dass mühelos Dienste wie Telefonie, Remote Diagnose, Notruf oder GPS unterstützen werden kann. Das Flea-System dient sowohl als Off-the-Shelf-Product für die Flottenausstattung als auch bei der Serienentwicklung eines Automobilsherstellers.

**Abbildung 1.1. Flea-System Spezifikation**

| Hardware | The Flea |
|---|---|
| Prozessor | 32bit Microprozessor |
| Arbeitsspeicher | 8-128 MB SDRAM |
| Flashspeicher | 32-1024 MB Flash |
| Ethernet | 1 x Fast Ethernet 100 Base T |
| GSM/GPRS | Quadband |
| Wireless LAN | IEEE802.11g (54MBit/s), optional |
| Bluetooth | vorhanden |
| GPS | 12 Kanal |
| Schnittstellen (extern) | 1 x RS232<br>1 x K-Line<br>4 x CAN<br>2 x USB |
| Eingangsspannung | 6 V bis 35V |
| Betriebstemperatur | -25 - +70 °C |
| Abmessungen | 160 mm x 30 mm x 100 mm |
| Gewicht | 0,8 kg |
| Software | |
| Betriebssystem | Embedded Linux, Kernel 2.4.x |
| Basisdienste | ssh, scp, thttp, dhcp, iptables, ppp, sms, webmin, hostap |
| Datenhaltung | Oracle Light, optional |
| Monitoring | HP Openview, optional |
| Kommunikation | VPN, Roaming, Resuming, optional |
| Anwendungen | Ortung, Tracking, Diagnose, optional |

## 1.1. Aufgabenstellung

Viele Technologien, die einst für große Enterprise Systeme entwickelt wurden, finden auch im Embedded Bereich ihre Berechtigung. Dazu gehören nicht nur Dinge wie virtuelle Speicherverwaltung und TCP/IP.

Ein aktueller Trend sind die Webservices in Embedded Systems. Dadurch sollen wertvolle Dienste wie automatisch generierte Wartungsanfragen, Ferndiagnose und automatische

Nachbestellung von Verbrauchsgütern möglich werden.

Ein Webservice ist dabei eine programmierbare Komponente, die einen bestimmten Dienst zur Verfügung stellt und über das Internet erreichbar ist. Webservices können in jeder Sprache entwickelt werden und erlauben den Zugriff über das sehr bekannte und Firewall freundliche HTTP Protokoll.

Ein Webservice soll Positionsdaten des Fahrzeugs, in dem das Fly System eingebaut ist, zur Verfügung stellen. Zugriff auf den Webservice geschieht über die GPRS Schnittstelle.

1. Evaluierung des Systems auf Tauglichkeit für Webservices
2. Konzeption und Design einer Webserviceplattform für das Flea-System. Festlegung der benötigten Applikationen, wie HTTP Server, SOAP und XML Funktionalität
3. Definition eines GPS Webservices, der vom System bereitgestellt werden kann
4. Implementierung eines Prototyp Webservice

# Kapitel 2. Analyse

Embedded-Systeme basieren zumeist auf der derselben Hardware wie Arbeitsplatzcomputer, sie unterliegen jedoch meist stark einschränkenden Randbedingungen wie relativ leistungsschwacher Prozessor, wenig Speicher und ein Betriebsysstem, das nur die notwendigsten Tools bzw. Programme zur Verfügung stellt.

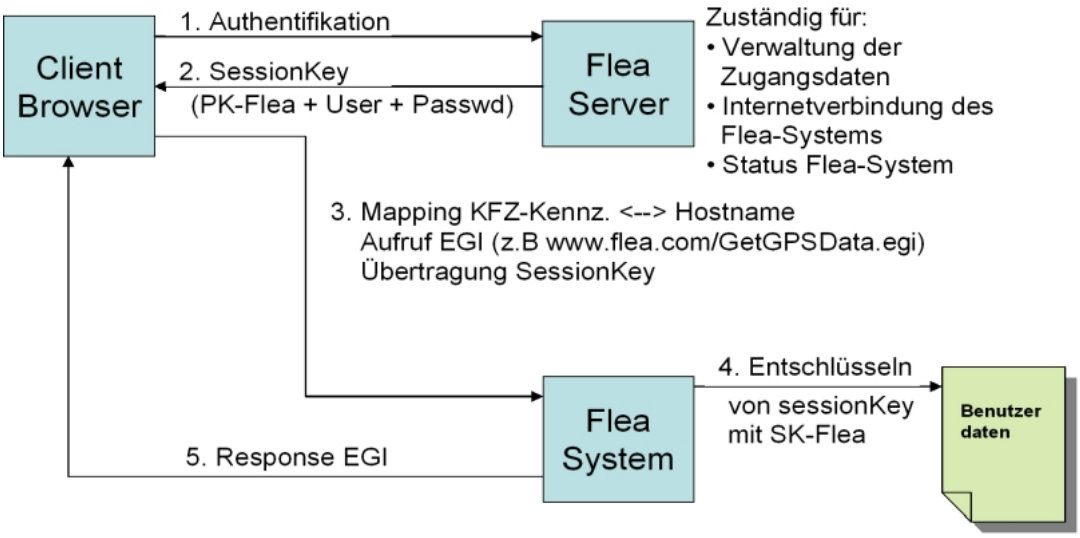
## 2.1. AppWeb

AppWeb der Fa. mbedthis ist ein schneller, kompakter, auf Standard basierter, speziell für embedded Systeme entwickelter Webserver. Der AppWeb Webserver hat einen sehr geringen CPU- und Speicherverbrauch (ca. 110K RAM und 1% CPU, bei minimaler Konfiguration) und ist sehr gut geeignet für embedded Systeme. Durch das Embedded Gateway Interface (EGI), welches ein effizienter Ersatz für das Common Gateway Interface (CGI) bietet, lassen sich auf Anfrage vom Client, Code innerhalb einer Applikation ausführen. EGI ist ein AppWeb Handler, der in C/C++ geschrieben wird und auf HTTP POST- und GET Requests antwortet. Während CGI immer einen externen Prozess startet, läuft EGI innerhalb des aufgerufenen Prozesses und das mit einem sehr kleinen Overhead. Dies bietet die Möglichkeit, Webservices ohne zusätzliche Tools/Programme wie SOAP, Skriptsprachen etc., die man beispielsweise für CGI bräuchte, welche zusätzlich für das embedded System portiert werden müssten, zu implementieren.

EGI ist ein AppWeb Handler, der in C/C++ geschrieben wird und auf HTTP POST- und GET Requests antwortet. Während CGI immer einen externen Prozess startet, läuft EGI innerhalb des aufgerufenen Prozesses und das mit einem sehr kleinen Overhead. Dies bietet die Möglichkeit, Webservices ohne zusätzliche Tools/Programme wie SOAP, Skriptsprachen etc., die man beispielsweise für CGI bräuchte, welche zusätzlich für das embedded System portiert werden müssten, zu implementieren.

# Kapitel 3. Design

**Abbildung 3.1. Aufruf GPS Webservice im Online Zustand (SSL)**



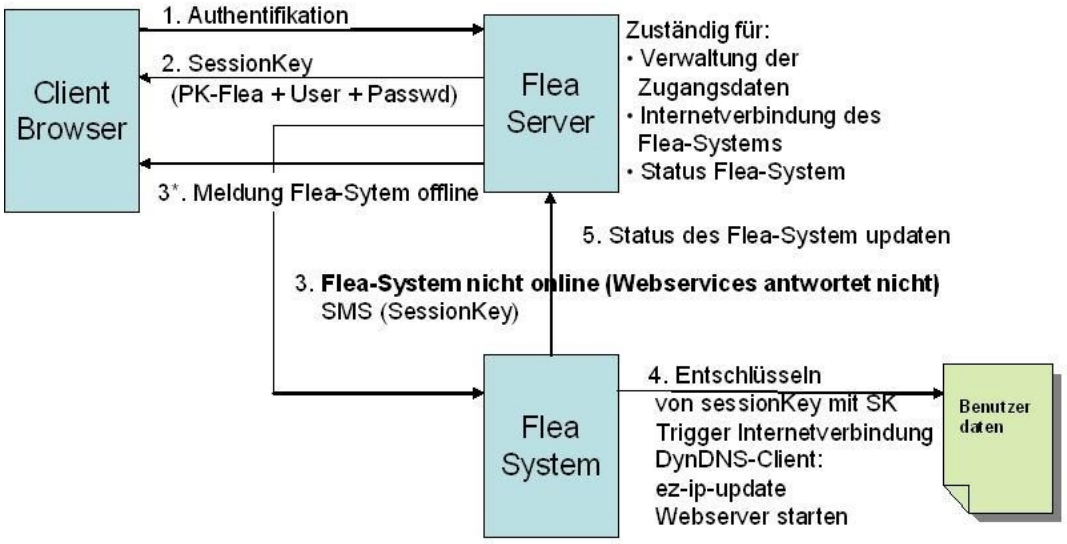

**Client:** Benutzer, der über einen Webbrowser das GPS-Webservice nutzt

**Flea-Server:** Zuständig für die Verwaltung der Zugangsdaten, enthält den Status und ist für die Triggerung der Internetverbindung des Flea-Systems zuständig.

**Flea-System:** Telematikeinheit der Fa. CarMediaLab GmbH

**Ablauf:** Bei der Nutzung des GPS-Webservices werden alle Daten verschlüsselt über SSL ausgetauscht. Im ersten Schritt muss sich der Benutzer am Flea-Server autorisieren. Bei erfolgreicher Anmeldung wird dem Client ein SessionKey übermittelt. Der SessionKey ist die Chiffre des Benutzernamen und Passwort mit dem öffentlichen Schlüssel des Flea-Systems. Anschließend gelangt der Benutzer auf eine Seite, die ihm die Möglichkeit bietet, das GPS-Webservice für das gewünschte KFZ durch die Eingabe des Kennzeichens zu nutzen. Das Mapping des Kennzeichens auf den Hostnamen des Flea-Systems erfolgt über eine XML-Datei. Dabei wird der SessionKey an das Flea-System übermittelt. Der SessionKey wird mit dem privaten Schlüssel dechiffriert und mit den lokalen Benutzerinformationen verglichen. Der Benutzer muss ebenfalls auf dem Flea-System registriert sein. Stimmen diese Informationen überein, bekommt der Benutzer eine Antwort vom Webservice.Die Nutzung des GPS-Webservices bei diesem Ablauf ist nur im Online Zustand des Flea-Systems möglich. Die Nutzung des GPS-Webservices im Offline Zustand folgt im

**Abbildung 3.2. Aufruf GPS Webservice im Offline Zustand (SSL)**




**Ablauf:** Ist das Flea-System nicht Online d.h. nicht über das Internet erreichbar, bekommt der Benutzer eine entsprechende Meldung und wird auf eine Seite weitergeleitet, auf der er den Status des Flea-Systems beobachten kann. Das Flea-System bekommt anschließend automatisch eine SMS mit der SessionKey und leitet den Internet-Trigger ein d.h. der SessionKey wird entpackt, dechiffriert und mit den lokalen Benutzerdaten verglichen. Stimmen diese überein, wird die Internetverbindung aufgebaut. Dabei bekommt es eine dynamische IP-Adresse, die mit dem DynDNS-Client (ez-ip-update) auf dem DynDNS-Server aktualisiert wird. Ein DynDNS- oder dynamischer Domain Name System-Eintrag bewirkt, dass ein Rechner, der eine wechselnde IP-Adresse besitzt, immer über den selben Namen angesprochen werden kann. Schließlich wird der Webserver auf Port 80 für alle verfügbaren Netwerkschnittstellen gestartet. Sieht der Benutzer den Online-Status des Flea-Systems, kann er den Webservice nutzen.

# Kapitel 4. Implementierung

## 4.1. Portierung AppWeb

Der AppWeb Webserver ist in C/C++ geschrieben. Für die Portierung wurde der Webserver mit der Toolchain der Fa. Adescom cross-compiliert. Die Toolchain besteht aus mehreren Komponenten und wird benötig um den ausführbaren Maschinen Code für die Tricore TC1130 Architektur zu erzeugen. Sie ist Architektur Abhängig wodurch jedes neue Prozessor Derivat desselben Models (TC1130) eine Änderung der Toolchain sehr wahrscheinlich zur Folge hat.

**Die Toolchain besteht im wesentlich aus:**

1. einem angepassten Compiler, der die „Mnemonics“ (Assembler Befehle) des Prozessors kennt und höhere Programmiersprachen dadurch übersetzen kann.
2. Bibliotheken und Header Dateien
3. Einer Angepassten Kernel Source
4. Eventuell angepasste Anwendungs Quellen

Generell mussten Makefiles angepasst werden um z.B. bestimmte Compiler-Optionen für den Tricore TC1130 Prozessor zu setzen. Es gibt bei einigen Projekten eine Datei Namens config.sub, die vom configure bei Crosscompilation ausgewertet wird. Sie kann einem das Crosscompilieren leichter machen, aber nur wenn die Hardwarevariante enthalten ist, was beim TC1130 nicht der Fall ist. Einige Stellen des Webservers mussten aufgrund der Architekturunterschiede oder fehlende Bibliotheken umgeschrieben werden.

## 4.2. Implementierung GPS-Webservices

Das GPS-System ist ein vom amerikanischen Verteidigungsministerium (DOD; Department of Defense) ersonnenes, realisiertes und betriebenes System, das aus (geplant) 24 Satelliten besteht (21 werden benötigt, 3 sind aktiver Ersatz; heute sind es allerdings meist um die 30 aktive Satelliten), welche die Erde in einer nominellen Höhe von 20200 km umkreisen. GPS Satelliten senden Signale aus, welche die genaue Ortsbestimmung eines GPS Empfängers ermöglichen. Die Empfänger können ihre Position ermitteln, wenn sie feststehend sind, sich auf der Erdoberfläche in der Erdatmosphäre oder in niederen Umlaufbahnen bewegen. GPS wird sowohl in der Luft-, Land- und Seefahrtnavigation als auch bei der Landvermessung und anderen Anwendungen eingesetzt, bei denen es auf genaue Positionsbestimmung ankommt. Das GPS-Signal wird jedem auf oder in der Nähe des Planeten kostenlos zur Verfügung gestellt, der einen GPS-Empfänger besitzt und eine

uneingeschränkte "Sicht" auf die Satelliten hat. Der eigentlich Name des Systems ist NAVSTAR (Navigation System for Timing and Ranging), bekannt ist es aber nur als GPS (Global Positioning System).

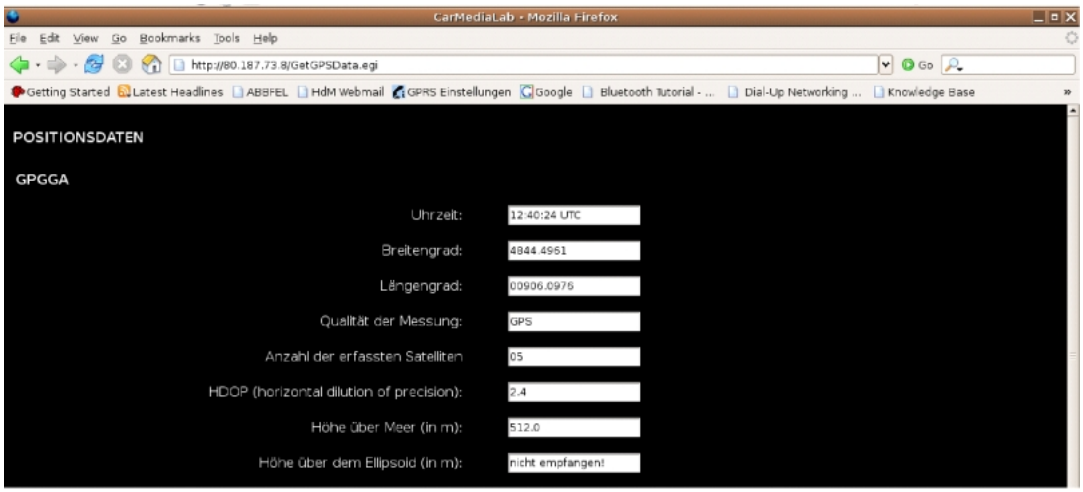
Die NMEA (National Marine Electronics Association, Nationale Vereinigung für Marineelektronik) engagiert sich für die Ausbildung und den Fortschritt der Marine-Elektronikindustrie und dem Markt, den diese bedient. Es handelt sich dabei um eine nicht auf Profit ausgelegte Vereinigung von Herstellern, Vertreibern, Ausbildungsinstitutionen und anderen mit Interesse an diesem Mark.

Mit Hilfe der weitestgehend standardisierten NMEA-Daten gelingt es sehr leicht, die Daten praktisch jedes GPS-Geräts mit einem Navigations- und Kartenprogramm auf dem PC, Laptop oder Handheld zu verwenden. Sogenannte GPS-Mäuse (GPS-Empfänger ohne Display nur mit serieller Schnittstelle) kommunizieren ausschliesslich auf diese Art mit Ihrer Aussenwelt. In der Seefahrt werden Kursplotter und ähnliches mit Hilfe von NMEA-Datensätzen mit Positionsdaten versorgt.

## 4.3. Webservice GetGPSData

Dieser Webservice hat die Aufgabe, die NMEA-Daten aus der GPS-Schnittstelle des Flea-Systems zu lesen, zu parsen und sie in einer vordefinierten HTML-Form zurückzugeben. Der Webservice ist nach dem EGI-Konzept des AppWeb Webservers in C/C++ implementiert. Kommt ein EGI-Request (GetGPSData.egi), ruft der AppWeb-Handler die entsprechende Methode (GetGPSData) auf, die weiteren Operationen wie z.B.: lesen des GPS-Interface, Parsen und Rückgabe der NMEA-Daten in HTML ausführt.

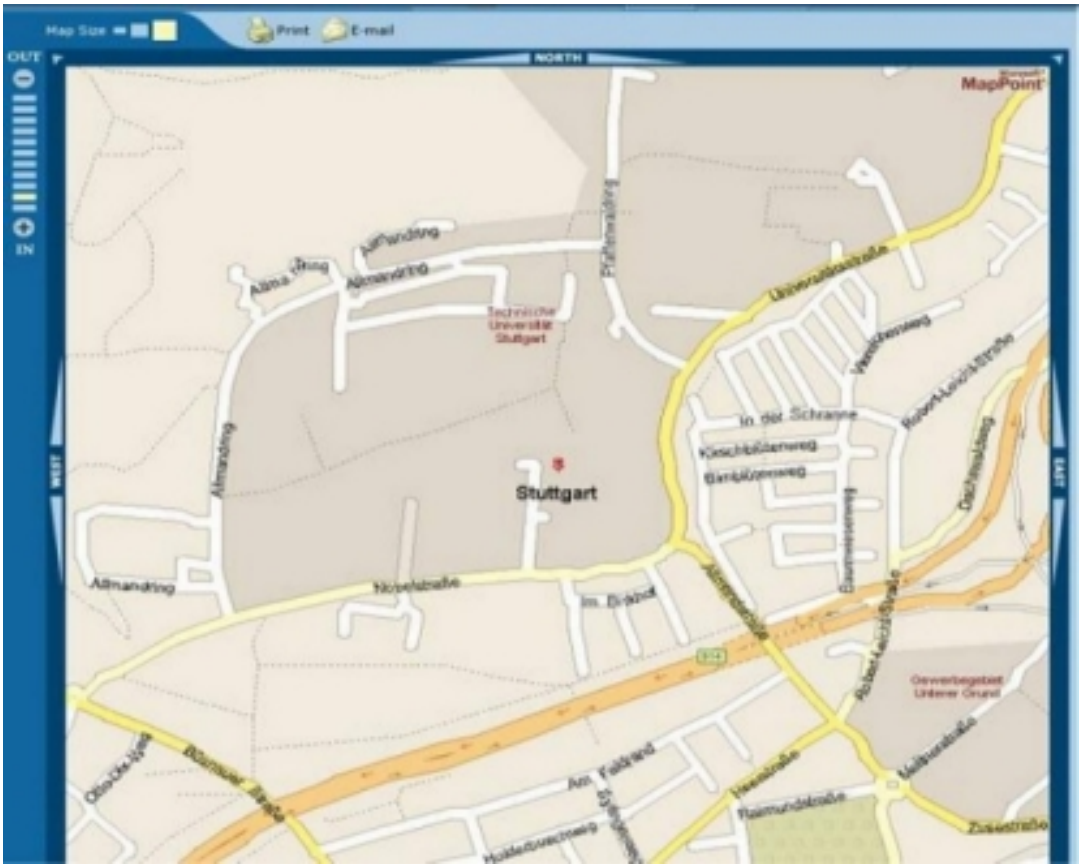
**Abbildung 4.1. GetGPSData.egi**



# 4.4. Webservice GetGPSMap

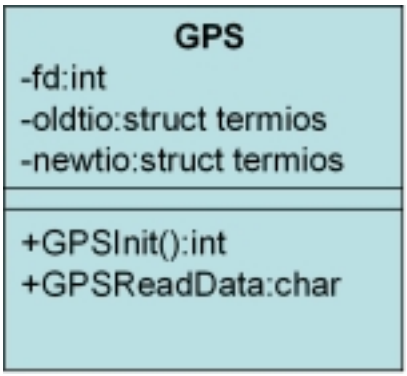
Dieser Webservice stellt mit Hilfe der NMEA-Daten die aktuelle Position des KFZ auf einer Landkarte dar. Der Webservice ist ebenfalls nach dem EGI-Konzept des AppWeb Webservers in C/C++ implementiert. Nach einer Umrechnung der Längen- und Breitengrade wird der Benutzer auf ein öffentliche MSN-Seite weitergeleitet, die die Position des KFZ auf einer Landkarte anzeigt.

**Abbildung 4.2. GetGPSMAP.egi**



# 4.5. GPS-Klasse

**Abbildung 4.3. GPS Class**

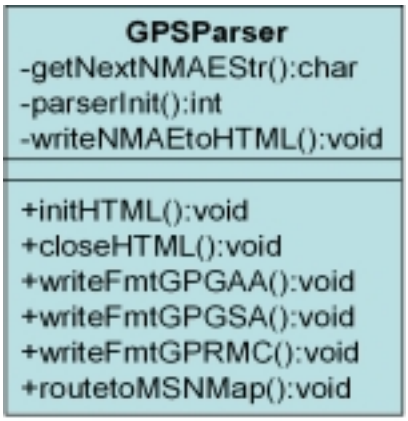



| Funktion | Beschreibung |
|---|---|
| public GPSInit() | Macht die notwendigen Einstellungen für die GPS-Schnittstelle wie z.B.: setzen der Portgeschwindigkeit, Hardwareflusskontrolle, von Startbit u. Stopbit, Parität usw.<br>Parameter: char *device<br>Return: 1 bei Fehler sonst 0 |

| Funktion | Beschreibung |
|---|---|
| public GPSReadData() | Ließt die rohen NMEA-Daten. Return: Anzahl der gelesenen Bytes<br>Parameter: char *buffer<br>Return: Anzahl der gelesenen Bytes |

# 4.6. GPSParser-Klasse

**Abbildung 4.4. GPSParser Class**




| Funktion | Beschreibung |
|---|---|
| private parserInit() | Vorbereiten des Parsers. NMEA-Buffer setzen.<br>Parameter: char *NMEA-Buffer<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| private getNextNMEAStr() | Iteriert durch den Kommagetrennten NMEA-Daten. Return: NMEA-Token<br>Parameter:<br>Return: NMEA-Token |

| Funktion | Beschreibung |
|---|---|
| private writeNMEAtoHTML() | Ausgabe des NMEA-Tokens in HTML-Form<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal char *head: HTML-Feldüberschrift char *input_name: HTML-Feldname char *input_value HTML-Feldwert<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public initHTML() | HTML-Formular vorbereiten (html-tag, body-tag usw. generieren)<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public closeHTML() | HTML-Formular schließen (body-tag, html-tag usw)<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kana<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public writeFmtGPGAA() | Formatieren von NMEA-GPGAA-Daten<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal char *NMEA-Buffer: NMEA-Buffer<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public writeFmtGP-GSA() | Formatieren von NMEA-GPGSA-Daten<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal char *NMEA-Buffer: NMEA-Buffer<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public writeFmtG-PRMC() | Formatieren von NMEA-GPRMC-Daten<br><br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal char *NMEA-Buffer: NMEA-Buffer<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| public routeMSNMap() | Berechnung der Längen und Breitengrade für MSN-Map. Weiterleiten auf die MSN-Map Seite<br>Parameter: *MaRequest: Zeiger auf den HTTP-Request-Kanal char *NMEA-Buffer: NMEA-Buffer<br>Return: void |

## 4.7. Log-In

Aus Sicherheitsgründen muss sich der registrierte Benutzer, bevor ein Webservice genutzt werden kann, am Flea-Server anmelden.

**dbconnect.php:** Die dbconnect.php ist in jede PHP-Seite per include integriert. Sie beinhaltet 5 Funktionen:

| Funktion | Beschreibung |
|---|---|
| connect() | Verbindungsaufbau zur MySQL-Datenbank |

| Funktion | Beschreibung |
|---|---|
| check_user() | Kontrolle ob für den angegebenen User mit dem betreffenden Passwort ein Eintrag in der Datenbank besteht.<br>Parameter: $name $pass<br>Return: UserId |

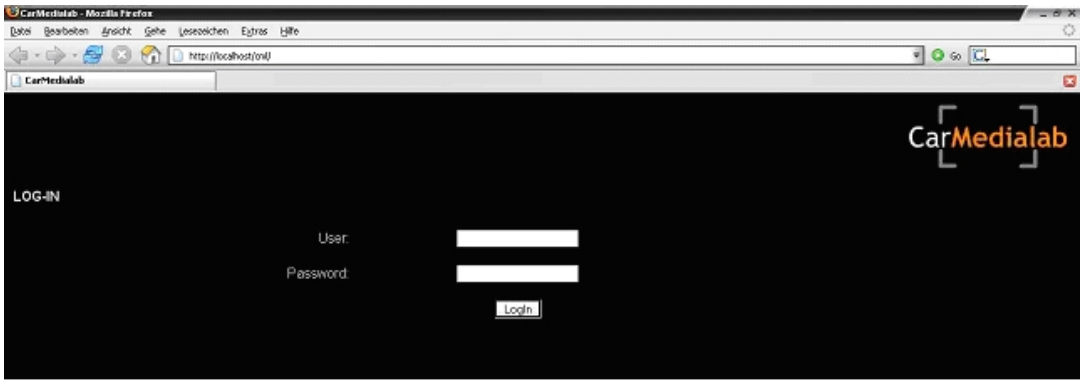
| Funktion | Beschreibung |
|---|---|
| login() | Zuteilung einer Session-Variable<br>Parameter: $userid<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| logged_in() | Kontrolle ob für den User momentan eine Session besteht<br>Parameter:<br>Return: void |

| Funktion | Beschreibung |
|---|---|
| logout() | Löscht die Session-Variable des Users aus der Datenbank<br>Parameter:<br>Return: void |

**index.php:** Die index.php ist die Startseite beim Einlogg-verfahren. Diese Seite hat zwei Zustände – entweder man ist noch nicht eingeloggt, dann wird die Log-In-Seite angezeigt, oder man ist schon eingeloggt, dann werden links zur Kennzeichen-Abfrage und zum Log-Out angezeigt. Nachdem man die korrekten Anmeldedaten eingegeben hat wird man auf die map.php weitergeleitet.

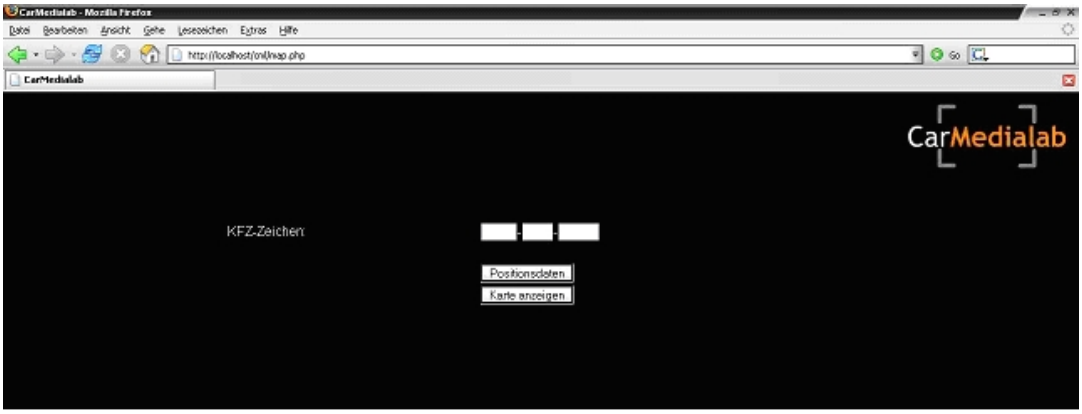
### Abbildung 4.5. Login



### Abbildung 4.6. Login processed


**map.php:** Diese Seite ist für die KFZ-Kennzeichen-Eingabe und sendet lediglich die Eingabe des Nutzers per HTTP_POST an die mapping.php

### Abbildung 4.7. Mapping KFZ <-> Hostname




**mapping.php:** Diese Datei beinhaltet einen XML-Parser, da die Kennzeichen mit der zugehörigen IP-Nummer des jeweiligen Flea-Systems in einer XML-Datei abgespeichert sind. Es wird für jeden Eintrag der XML-Datei das eingegebene Kennzeichen verglichen, und wenn es identisch ist, wird der Nutzer auf den Web-Server des Flea-Systems im Auto weitergeleitet. Dort wird das entsprechende EGI-Script aufgerufen.

# Anhang A.

# Glossar

| | |
|---|---|
| EGI | Embedded Gateway Interface |
| CGI | Common Gateway Interface |
| SSL | Secure Socket Layer |
| XML | Extensible Makro Language |
| SOAP | Simple Object Access Protocol |
| GSM | Global System for Mobile Communications |
| GPRS | General Packet Radio Service |
| HTTP | Hypertext Transfer Protocol |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| CPU | Central Processing Unit |
| PK/SK | Private Key/Secret Key |
| KFZ | Kraftfahrzeug |
| SMS | Short Message Service |
| DynDNS | Dynamisches Domain Name System |
| NMEA | National Marine Electronics Association |
| NMEA | National Marine Electronics Association |
| MSN | Microsoft Network |
| PHP | Hypertext Preprocessor |

# Index