

# Studienarbeit ISO 15118

---

Von Alexander Barth & Michael Röhrich

## Inhaltsverzeichnis

<b>1 Aufgabenstellung</b>	<b>3</b>
1.1 Aufgabenstellung des Projektes	3
1.2 Warum gibt es die ISO 15118	3
1.3 Ablauf des Projekts	4
<b>2 Programme</b>	<b>6</b>
2.1 Mindjet	6
2.2 Eclipse	6
<b>3 Der ISO 15118 Code</b>	<b>9</b>
3.1 Vorgehensweise der Einarbeitung	9
<b>4 Verwendung der ISO-Normen</b>	<b>12</b>
4.1 ISO 15118 Teil 1	12
4.1.1 Verwenden des Stacks und vorhandene Funktionen	12
4.1.2 Akteure der Kommunikation	12
4.1.3 Kommunikationsablauf	13
4.1.4 Erläuterung Annex A	14
4.1.5 Erläuterung Annex B	15
4.1.6 Erläuterung Annex C (Ladeszenarien)	15
4.2 ISO 2:	15
4.2.1 Sicherheit	15
4.2.2 PnC & EIM	16
4.2.3 Benötigte Signale der Ladearten	17
4.2.4 Response-Nachrichten	19
4.2.5 Zustandsautomaten	20
4.2.6 Ablauf Diagramm der Kommunikation	23
4.2.7 Annex A	25
4.2.8 Weitere Anhänge	25

---

<b>5 Strukturübersicht der Botschaften</b>	<b>26</b>
5.1 Aufbau des Dokuments	26
<b>6 Programmbeispiel</b>	<b>28</b>

ABILDUNG 1: UMGEBUNGSVARIABLEN SETZEN .....	7
ABILDUNG 2: ECLIPSE EINSTELLUNGEN 1 .....	8
ABILDUNG 3: ECLIPSE EINSTELLUNGEN 2 .....	8
ABILDUNG 4: AKTEURE DER KOMMUNIKATION (SIEHE (1), SEITE 14) .....	12
ABILDUNG 5: USE CASE ELEMENTS (SIEHE (1), SEITE 16).....	13
ABILDUNG 6: UNTERSCHRITTE DER USE CASES (SIEHE (1), SEITE 16).....	14
ABILDUNG 7: EIM & PNC (SIEHE (2), SEITE 116).....	16
ABILDUNG 8: BENÖTIGE SIGNALE (SIEHE (2), SEITE 117) .....	17
ABILDUNG 9: ZEITLICHES VERHALTEN VON BOTSCHAFTEN (SIEHE (2), SEITE 145).....	18
ABILDUNG 10: RESPONSE CODES (SIEHE (2), SEITE 153) .....	19
ABILDUNG 11: ZUSTANDSAUTOMAT EVCC (SIEHE (2), SEITE 159) .....	21
ABILDUNG 12: ZUSTANDSAUTOMAT SECC (SIEHE (2), SEITE 167) .....	22
ABILDUNG 13: ABLAUFDIAGRAMM AC,EIM (TEIL 1), (SIEHE (2), SEITE 171).....	23
ABILDUNG 14: ABLAUFDIAGRAMM AC,EIM (TEIL 2), (SIEHE (2), SEITE 172).....	24
ABILDUNG 15: NACHRICHTEN UND USE CASES (SIEHE (2), SEITE 180).....	25
ABILDUNG 16: UML- CHART GESAMT (MINDJET).....	27
ABILDUNG 17: BEISPIEL- BOTSCHAFT AUS UML- CHART (MINDJET) .....	28
ABILDUNG 18: BEISPIELPROGRAMM BILD 1 .....	29
ABILDUNG 19: BEISPIELPROGRAMM BILD 2 .....	FEHLER! TEXTMARKE NICHT DEFINIERT.
ABILDUNG 20: BEISPIELPROGRAMM BILD 3 .....	31
ABILDUNG 21: BEISPIELPROGRAMM BILD 4 .....	31
ABILDUNG 22: BEISPIELPROGRAMM BILD 5 .....	33
ABILDUNG 23: BEISPIELPROGRAMM BILD 6 .....	34
ABILDUNG 24: BEISPIELPROGRAMM BILD 7 .....	34
ABILDUNG 25: BEISPIELPROGRAMM BILD 8 .....	35
ABILDUNG 26: BEISPIELPROGRAMM BILD 9 .....	36
ABILDUNG 27: BEISPIELPROGRAMM BILD 10 .....	36
ABILDUNG 28: BEISPIELPROGRAMM BILD 11 .....	37

# 1 Aufgabenstellung

## 1.1 Aufgabenstellung des Projektes

Ziel der Arbeit:

Der Ablauf der Kommunikation zwischen Elektrofahrzeugen und der Ladesäule soll dargestellt werden. Mit dem bereits vorhandenen Kommunikationsstack soll ein Programm geschrieben werden, mit Hilfe dessen die Kommunikation zwischen den beteiligten Geräten aufgezeigt werden kann. Die realen Funktionen sollen angezeigt werden, indem die zwei geschriebenen Programme auf verschiedenen Rechnern parallel ablaufen. Hierbei läuft auf dem einen Rechner das Programm des Fahrzeugs und auf dem anderen Rechner das Programm der Ladesäule ab. Die Rechner sollen über eine frei wählbare Schnittstelle kommunizieren. Des Weiteren war als Kann-Ziel angedacht, dass das Programm danach auf das Rechnersystem Raspberry Pi aufgespielt werden soll um es mit diesem zu simulieren.

## 1.2 Warum gibt es die ISO 15118

In der derzeitig anlaufenden Energiewende soll unter anderem auch ein intelligentes Stromnetz (smart grid) erschaffen werden. Hierzu ist eine optimale Vernetzung zwischen Stromerzeugern, Stromspeichern sowie den elektrischen Verbrauchen nötig. In der Branche wird davon gesprochen, dass Elektroautos im besten Falle sogar eine große Rolle in diesem System spielen könnten. Grund hierfür ist, dass Elektroautos sowohl als steuerbare Verbraucher (die nur Strom beziehen, wenn viel erneuerbare Energie verfügbar ist), sowohl als Akkumulatoren zur Speicherung der überschüssigen Energie und sogar als Erzeugungseinheit (Rückspeisung/Einspeisung von Blindleistung) betrachtet werden können.

Um diesen Anforderungen nachkommen zu können, bedarf es eines leistungsfähigen Kommunikationsprotokolls, mit welchem die Elektroautos mit den Ladesäulen und der dahinter liegenden Infrastruktur bzw. den Rechnersystemen kommunizieren. Ebenfalls wird in Zukunft das Thema Sicherheit weiter eine große Rolle spielen, deshalb ist als Basis für die Kommunikation eine abhörsichere und verschlüsselte

Kommunikation notwendig. Für die Standardisierung der Kommunikation wurde die achtteilige Norm ISO/IEC 15118 entwickelt.

Der Kommunikationsstack der ISO kann unter der Seite <http://openv2g.sourceforge.net> heruntergeladen werden.

### 1.3 Ablauf des Projekts

Projektmanagement:

Zu Beginn des Projektmanagements stellten wir für die einzelnen Arbeitsschritte je ein Arbeitspaket auf. Mit diesen Arbeitsschritten planten wir das Projekt abschließen zu können. Zuerst war geplant, dass es eine Einarbeitungszeit zur Einarbeitung in die ISO 15118 (ISO 15118 Teil 1) & (ISO 15118 Teil 2), sowie den Kommunikationsstack gibt. Des Weiteren sollte eine Einarbeitung in Linux, sowie in das Rechnersystem Raspberry Pi erfolgen.

Anpassung der Ziele:

In einer folgenden Besprechung mit dem Auftraggeber wurde klar, dass wir die oben genannten Ziele nicht alle erreichen werden. Es zeigte sich, dass schon die Einarbeitung sehr viel Zeit in Anspruch nehmen würde. Deswegen wurde vereinbart, dass wir uns primär in die ISO 15118 einarbeiten. Als Ergebnis dieser Arbeit sollte die ISO 15118 erklärt werden mit dem Verweis auf die dazu benötigten Seiten innerhalb der Norm um ein späteres Programmieren zu erleichtern. Zusätzlich soll das vorhandene Beispielprojekt innerhalb der ISO kurz erklärt werden.

Anpassung der Prioritäten – neue Vorgehensweise:

Wie schon erwähnt stellte sich im Laufe der Einarbeitung in die ISO 15118 heraus, dass selbst die Einarbeitung einen erheblichen Zeitaufwand darstellt. Unter anderem, weil sich die Dokumentation der ISO auf mehrere Hundert Seiten verteilt. Wir haben festgestellt, dass die Struktur des Kommunikationsstacks sehr aufwendig ist.

Deshalb haben wir uns entschlossen ein Baumdiagramm / UML Darstellung zu erstellen, indem aufgezeigt wird, welche Botschaften und Signale zwischen dem Elektrofahrzeug und der Ladestation benötigt werden. Die einzelnen

## Automotive Systems Engineering

Kommunikationsschritte und ihre Unterpunkte werden hier graphisch untereinander angeordnet. Zu den einzelnen Schritten wurde jeweils ein beschreibender Kommentar eingefügt. Neben der Beschreibung werden in diesem Kommentar auch gleich die Dimensionen der Nachricht festgehalten. Das heißt, es wird notiert, in welcher Form, z.B. String, Char, oder Ähnliches die Variable erwartet wird.

## 2 Programme

### 2.1 Mindjet

Mindjet ist eine Software mit der man Mindmaps erstellen kann. Mindmaps sind graphische Darstellungen mit Kästchen und Pfeilen um komplizierten Strukturen ordentlich darstellen zu können. Da die in der ISO Teil 2 verwendete Struktur der Botschaften sehr über viele Seiten verteilt ist, wurde der Botschaftsaufbau in einer Mindmap nachgebildet. Hierin sind alle Botschaften auf einen Blick ersichtlich und der Umfang der Komplexität wird deutlich sichtbar. Die Beschreibung jedes einzelnen Elementes wurde aus der ISO herausgesucht und als „Notiz“ im Mindmap hinzugefügt.

In Mindjet stehen Import- und Export Funktionen zur Verfügung, in der z.B. die Mindmap als Word- Dokument exportiert werden kann oder beliebige Word-Dokumente importiert werden können.

### 2.2 Eclipse

Eclipse wurde verwendet um den C-Code der ISO 15118 zu kompilieren. Hierbei gab es jedoch einige Probleme. Es war mühsam die Einstellungen so hinzubekommen, dass der Code lauffähig war. Im Folgenden werden die wichtigsten Einstellungen genannt, die getätigt werden müssen, damit das Kompilieren des Codes in Eclipse möglich ist.

#### Die Richtige Installation von Eclipse:

Im Internet wird nach „Eclipse for C and C++“ gesucht. Bei der Installation muss die richtige Variante ausgewählt werden, z.B. für Windows 64 Bit oder MAC.

Nachdem die C und C++ Installationsdateien herunter geladen wurden, ist diese im zip- Format komprimiert. Diese sollte auf das C Laufwerk entpackt werden. Hierzu sollte auf (C:) ein Ordner mit einem möglichst kurzen Ordnernamen angelegt werden, z.B. „Eclipse“.

Zudem sollte das JDK(Java Development Kit) neu installiert werden. Zusätzlich muss der neuste MinGW-Compiler installiert sein.

Im Anschluss daran sollte man in den Systemeinstellungen folgende Änderungen vornehmen:

In den Systemeigenschaften muss die Umgebungsvariable „Path“ neu gesetzt werden.

#### Vorgehen bei Windows 8:

Systemsteuerung -> System und Sicherheit -> System -> Erweiterte Einstellungen ->

Umgebungsvariablen -> Neu

Dann bei „Name der Variablen“: „Path“ eingeben.

Und bei „Wert der Variablen“ den Link zum MinGW-Compiler hineinkopieren.

#### Beispiel:

C:\ProgramData\Oracle\Java\javapath;C:\MinGW\bin

Die Vorgehensweisen können der Abbildung 1 entnommen werden.

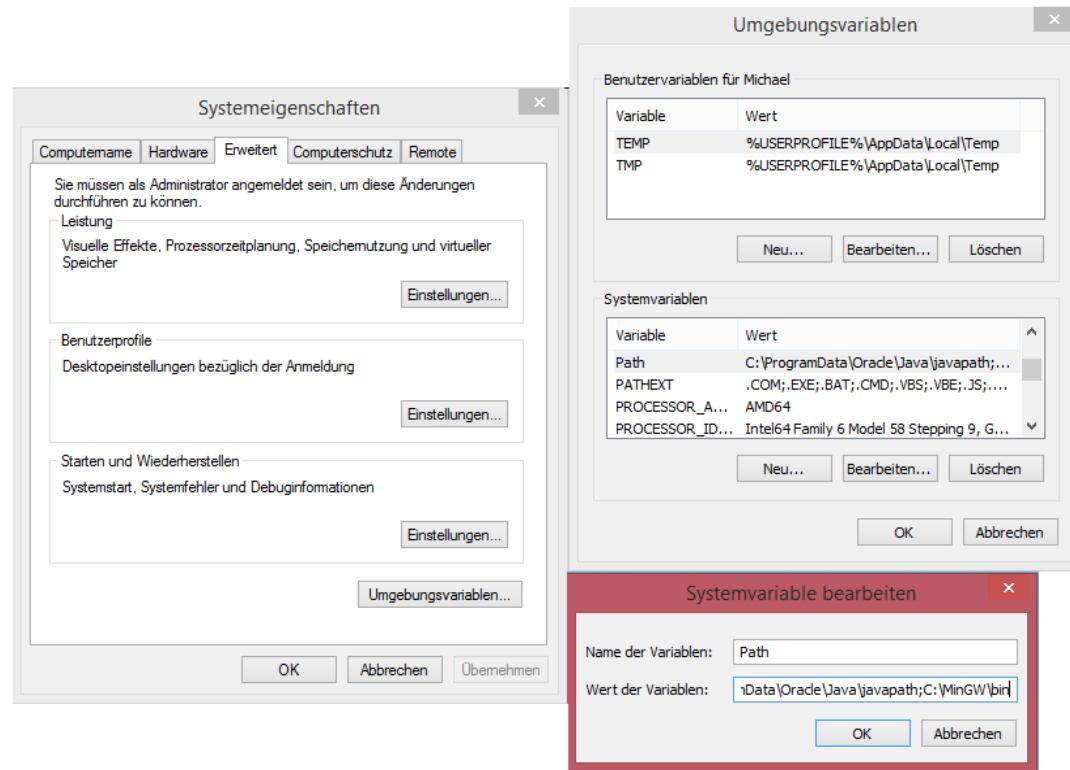
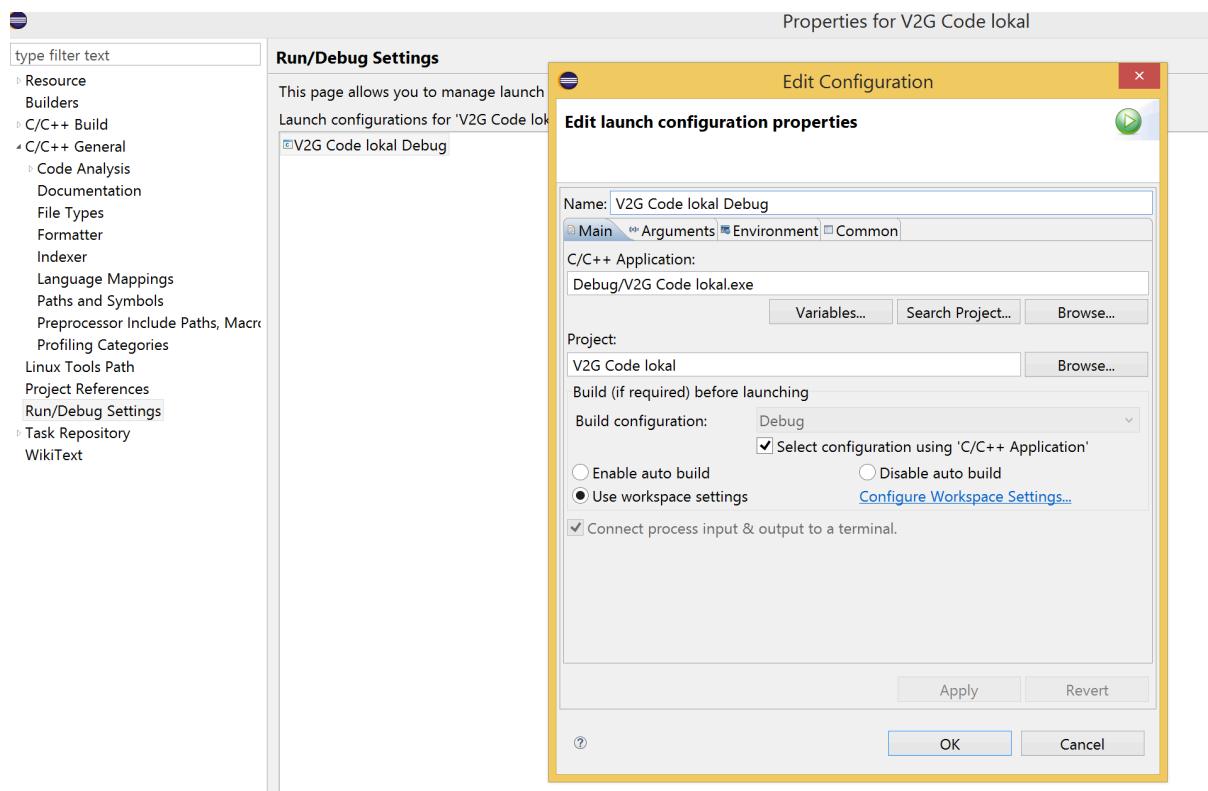
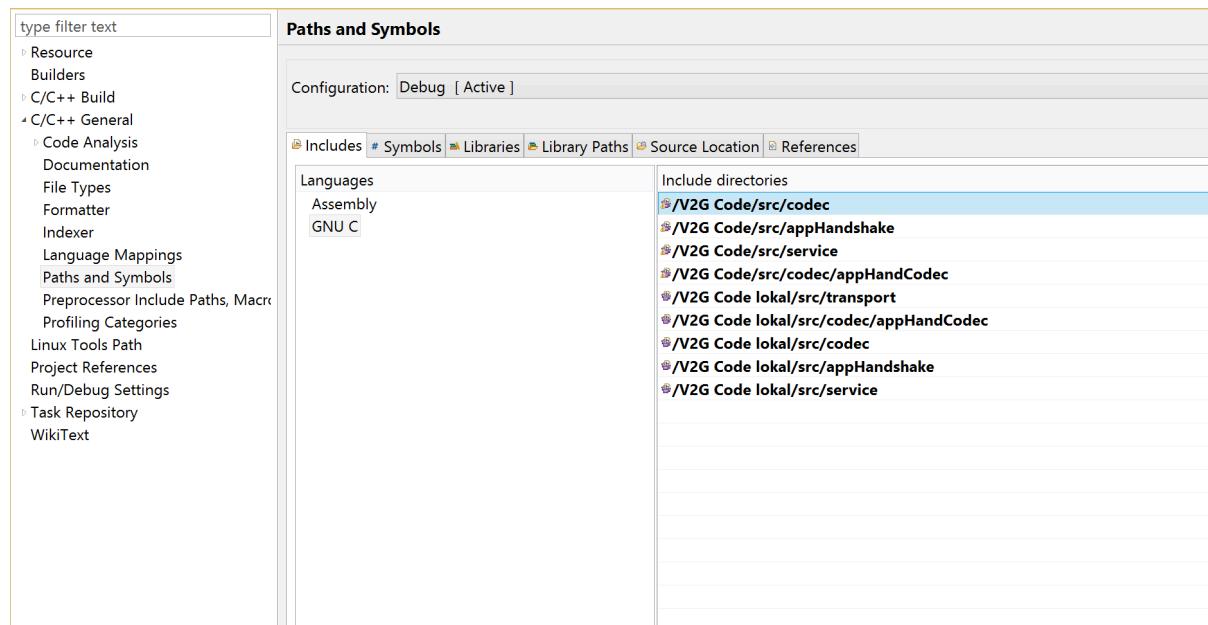


Abbildung 1: Umgebungsvariablen setzen

Nachdem nun Eclipse installiert und die Software der ISO15118 importiert wurde müssen noch Einstellungen in den Eigenschaften (Properties) vorgenommen werden. Diese Einstellungen können der Abbildung 2 und der Abbildung 3 entnommen werden.


**Abbildung 2: Eclipse Einstellungen 1**

**Abbildung 3: Eclipse Einstellungen 2**

### 3 Der ISO 15118 Code

#### 3.1 Vorgehensweise der Einarbeitung

Zuerst haben wir den ersten Teil der ISO 15118 durchgelesen, der allgemeine Informationen enthält. Anschließend folgte Teil 2. Aus Teil 2 wird die Struktur der Kommunikation entnommen.

Zu Allererst sollte man sich mit den Fachbegriffen und ein paar Abkürzungen vertraut machen um die Erklärungen in der ISO verstehen zu können. Im Folgenden haben wir die wichtigsten Begriffe erklärt, die nötig waren um sich in der ISO zurechtzufinden.

##### Authentication:

Die Authentifizierung ist ein Vorgang zwischen dem EVCC(Elektrofahrzeugs-Kommunikationsregler) und dem SECC(Versorgungs-Ausrüstungs-Kommunikationsregler) oder zwischen dem Benutzer und dem EVSE(Elektrofahrzeugs- Versorgungsausrüstung). Hierbei werden die Nachrichten auf Gültigkeit / Richtigkeit überprüft.

##### Certificate:

Das Zertifikat ist ein elektronisches Dokument, welches eine digitale Unterschrift verwendet um einen öffentlichen Schlüssel mit einer Identität zu binden.

##### Charging scenario:

Ein Ladeszenario ist eine Kombination von Use-Case Elementen(Möglichkeiten von Zuständen von Komponenten).

##### Contract ID:

Die Vertrags-Identifikationsnummer ist eine Nummer, die zur Identifikation eines Vorgangs zwischen den Parteien führt. Die enthält Informationen über die einzelnen Komponenten. Die ID kann fahrzeugspezifisch oder kundenspezifisch sein. Der Kunde kann in diesem Fall der Fahrer oder der Eigentümer des Fahrzeugs sein.

Energy Transfer Type:

Ist ein Element, das dem Elektrofahrzeug (EV) erlaubt seinen gewünschten Energieübertragungstyp im Fall, dass der EVSE diesen enthält, auszuwählen. Unterstützt werden diverse Möglichkeiten des Ladens, sowie diverse Stecker und Steckdosen gemäß IEC 62196.

Identification:

Ist ein Verfahren um produktspezifische Informationen zwischen Fahrer und EVCC auszutauschen. Der Fahrer hat hier die Möglichkeit z.B. Daten für die Zahlungsmodalitäten, wie z.B. die Kreditkartennummer zu hinterlegen. Im Gegenzug hierzu bekommt der Fahrer Informationen über die technischen Eigenschaften des Systems.

Primary Actor:

Hauptakteure die direkt am Ladeprozess beteiligt sind.

Sales Tariff Table:

Diese Tabelle enthält die aktuellen Preise für den von der Ladesäule zur Verfügung gestellten Strom. Diese Tabelle zeigt Angebot und Nachfrage auf, um evtl. einen geschickten Zeitpunkt zum Aufladen zu wählen, wenn z.B. die Windkraftanlagen oder Photovoltaikanlagen ein Maximum an Energie produzieren.

Secondary Actor:

Hierbei handelt es sich um Nebenakteure die am Ladeprozess beteiligt sein können, ob tatsächlich eine Beteiligung vorhanden ist, kann auch vom Use-Case abhängig sein.

Use Case:

Beschreibt einen Gebrauchs-Fall, indem aufgezeigt wird, „wer“ etwas mit dem gefragten System und „was“ mit dem gefragten System angefangen werden kann.

Die hier genannten Fachbegriffe sind nur die von uns am meisten verwendeten, falls weitere Fachbegriffe benötigt werden, so sind diese in der ISO Teil 1, auf Seite 2-9 zu finden.

Abkürzungen:

Im Folgenden sollen die wichtigsten Abkürzungen erklärt werden, die nötig waren um sich in der ISO zurechtzufinden.

**EP:** Electricity Provider => Energieversorger

**EV:** Electro Vehicle => Elektrofahrzeug

**EVCC:** Electric Vehicle Communication Controller => Elektrofahrzeugs Kommunikationsregler

**EVSE:** Electric Vehicle Supply Equipment => Elektrofahrzeugs Versorgungsausrüstung

**ECU:** Electronic Control Unit => Elektronisches Steuergerät

**EIM:** External Identification Means => Externes Identifikations-Behelfsmittel

**MO:** Meter Operator => Messdienst

**OEM:** Original Equipment Manufacturer => Erstausrüster / Originalhersteller

**PnC:** Plug and Charge => Einsticken und Aufladen

**SECC:** Supply Equipment Communication Controller => Versorgungsausrüstungs-Kommunikationsregler

**V2G:** Vehicle to Grid => Fahrzeug zu Verkehrsnetz

Die hier genannten Abkürzungen sind nur die von uns am meisten verwendeten, falls weitere Abkürzungen benötigt werden, so sind diese in der ISO Teil 1, auf Seite 9&10 zu finden.

Auf den Seiten 3 bis 6 der ISO 2 sind Erläuterungen von Fachbegriffen und Aufschlüsselungen zu Abkürzungen zu finden. Diese wurden bei der bisherigen Arbeit nicht wesentlich benötigt, können aber für künftige Arbeiten relevant werden.

## 4 Verwendung der ISO-Normen

### 4.1 ISO 15118 Teil 1

#### 4.1.1 Verwenden des Stacks und vorhandene Funktionen

#### 4.1.2 Akteure der Kommunikation

Die folgende Darstellung zeigt die wichtigsten Akteure bei der Kommunikation zwischen Elektrofahrzeug und Ladesäule auf. Hauptakteure sind das Fahrzeug, die Ladesäule und der Fahrzeughalter. In der Darstellung sind dem Fahrzeug die Systemteile Ladegerät, Steuergerät und Mensch-Maschinen-Schnittstelle untergeordnet. Beim Elektrofahrzeug sind unter anderem der Stromzähler, die Verwaltung der Bezahlvorgänge und die elektrischen Kontakte wichtige Unterfunktionen.

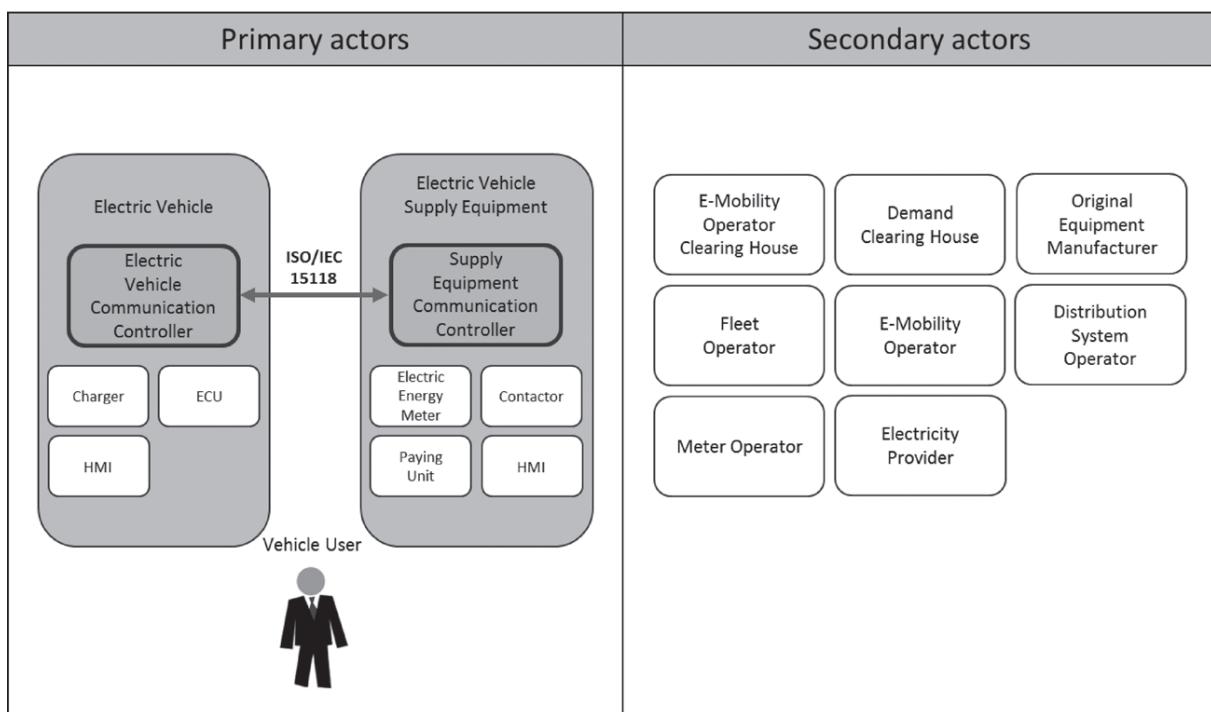
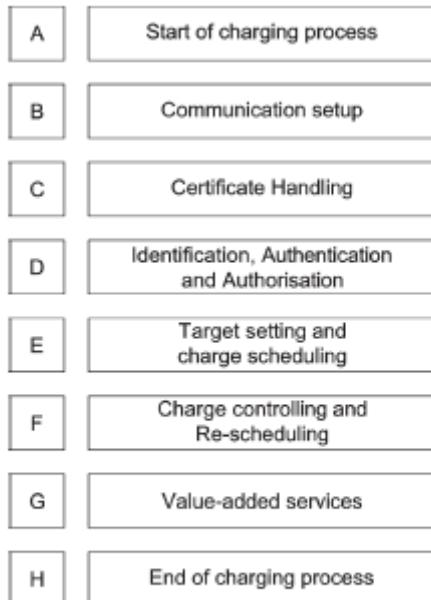


Abbildung 4: Akteure der Kommunikation (Siehe (ISO 15118 Teil 1), Seite 14)

#### 4.1.3 Kommunikationsablauf



**Figure 2 — Use case elements function groups**

**Abbildung 5: Use case elements (Siehe (ISO 15118 Teil 1), Seite 16)**

Der komplette Ablauf der Kommunikation zwischen dem Fahrzeug und der Ladesäule vom Beginn bis zum Ende eines Ladevorgangs ist in acht funktionelle Gruppen unterteilt. Jeder Kommunikationsablauf muss alle acht Schritte enthalten, jedoch können die Schritte intern voneinander variieren. Die Schritte sind in der ISO 1 auf Seite 15 kurz aber deutlich erklärt, deswegen wird hier nicht näher darauf eingegangen.

Table 1 — Overview of elements of use cases

No.	Use case element name / grouping
A1	Begin of charging process with forced High Level Communication
A2	Begin of charging process with concurrent IEC 61851-1 and High Level Communication
B1	EVCC/SECC communication setup
C1	Certificate update
C2	Certificate installation
D1	Authorization using Contract Certificates performed at the EVSE
D2	Authorization using Contract Certificates performed with help of SA
D3	Authorization at EVSE using external credentials performed at the EVSE
D4	Authorization at EVSE using external credentials performed with help of SA
E1	AC charging with load levelling based on High Level Communication
E2	Optimized charging with scheduling to secondary actor
E3	Optimized charging with scheduling at EV
E4	DC charging with load levelling based on High Level Communication
E5	Resume to Authorized Charge Schedule
F0	Charging loop
F1	Charging loop with metering information exchange
F2	Charging loop with interrupt from the SECC
F3	Charging loop with interrupt from the EVCC or user
F4	Reactive power compensation
F5	Vehicle to grid support
G1	Value added services
G2	Charging details
H1	End of charging process

Abbildung 6: Unterschritte der Use cases (Siehe (ISO 15118 Teil 1), Seite 16)

In obenstehender Tabelle sind alle vorhandenen Unterschritte der acht Gruppen notiert. Auf den Seiten 18-44 sind diese genau erklärt.

#### 4.1.4 Erläuterung Annex A

Im Anhang Annex A der ISO15118-Teil1 wird der generelle Aufbau der Infrastruktur aufgezeigt. Hierbei werden die verschiedenen Möglichkeiten auf welchen

Kommunikationsebenen kommuniziert wird. Des Weiteren wird darauf eingegangen, dass der Aufbau der Kommunikation zwischen einer Ladesäule und einem oder mehreren Fahrzeugen auf mehrere Weisen ausgeführt werden kann. Auf der Abbildung auf Seite 49 ist zu sehen, dass der Aufwand beim gleichzeitigen Laden von mehreren Fahrzeugen weitaus größer ist, weil die Fahrzeuge in Summe einen großen Strom ziehen und somit der fließende Strom auf die Fahrzeuge aufgeteilt werden muss.

#### 4.1.5 Erläuterung Annex B

Im Anhang Annex B der ISO15118-Teil1 wird das Thema Sicherheit abgehandelt. Hierbei werden die Ansprüche an die Sicherheit der Kommunikation dargestellt.

#### 4.1.6 Erläuterung Annex C (Ladeszenarien)

Im Anhang Annex C wird anhand von Anwendungsbeispielen die Anwendung der verschiedenen Use-Case Elementen erklärt. In verschiedenen Strukturbäumen wird aufgezeigt welche möglichen Pfade bei den unterschiedlichen Szenarien durchlaufen werden können. Hierbei kann es sein, dass manche Teil-Anwendungen nicht verwendet werden können, diese sind dann in der ISO geschwärzt. Ein Beispiel hierzu ist die Darstellung auf Seite S.62, hierbei handelt es sich um einen Ladevorgang an einer privaten Steckdose. Hierbei entfallen E2-E4, da diese bei diesem Ladeszenario nicht angewendet werden können.

### 4.2 ISO 2:

#### 4.2.1 Sicherheit

Ab Seite 10 ist eine Beschreibung der verschiedenen Arten der Sicherheitsübertragung zu finden.

Die Darstellungen zeigen die online- und die semi-online Kommunikation. Hierbei werden nur die sicherheitsrelevanten Stellen der Strukturen abgehandelt.

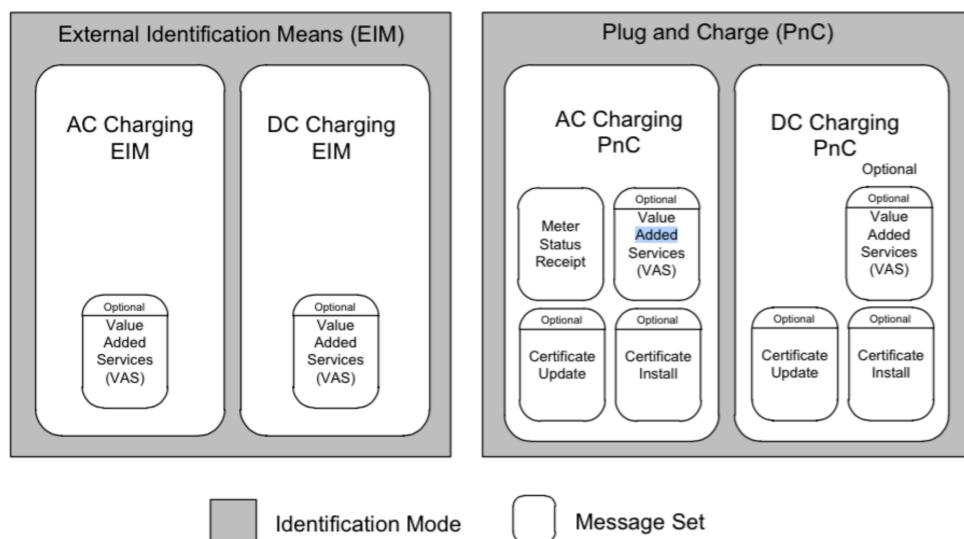
Die Tabelle auf Seite 14 zeigt den Inhalt und den Aufbau eines Zertifikats

Auf Seite 26 wird das V2G Transport-Protokoll erklärt. Es dient dazu die Nachrichten zwischen zwei Instanzen / Parteien, welche mit diesen arbeiten können, zu übertragen.

#### 4.2.2 PnC & EIM

Auf Seite 114 wird der Unterschied zwischen PnC und EIM genauer erklärt. Dieser kann der Abbildung 7 entnommen werden.

**ISO/IEC DIS 15118-2**



**Figure 91 — Overview on Identification modes and Message Sets**

Abbildung 7: EIM & PnC (Siehe (ISO 15118 Teil 2), Seite 116)

#### 4.2.3 Benötigte Signale der Ladearten

In den Tabellen auf den Seite 117 bis 134 kann entnommen werden, welche Signale bei welcher Art des Ladevorgangs bei den verschiedenen Ladevorgängen „gesendet werden“ müssen, „optional zu senden“ sind oder „nicht gesendet werden“ müssen.

Auf den Seiten 135 bis 141 sind noch Informationen zu den einzelnen Message-Sets aus den Tabellen zu finden. Des Weiteren werden die einzelnen Lademöglichkeiten beschrieben.

**ISO/IEC DIS 15118-2**

**[V2G2-668]** An SECC shall support the processing of a parameter if it is marked with an "M" for the SECC in a request message.

**Table 98 — Mandatory messages and message elements of Message Sets**

V2G Message							Message Set																	
Name	Parameter Level						EVCC	AC Charging Elm	EVCC	DC Charging Elm	EVCC	AC Charging PnC	EVCC	DC Charging PnC	EVCC	Option: Certificate Update	EVCC	Option: Certificate Installation	EVCC	Options: MeteringReceipt	EVCC	Option: VAS	EVCC	
	1	2	3	4	5	6	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	SECC	
Supported App Protocol Req	ProtocolNameSpace						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	VersionNumberMajor						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	VersionNumberMinor						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	SchemalD						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	Priority						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	Supported App Protocol Res						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
Session Setup Req	Response Code						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	SchemalD						M	O	M	O	M	O	M	O	M	-	-	-	-	-	-	-	-	-
	EVCCID						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	EVSEID						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
Session Setup Res	Response Code						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	EVSEID						M	M	M	M	M	M	M	M	M	-	-	-	-	-	-	-	-	-
	Datetime Now						O	O	O	O	O	O	O	O	O	-	-	-	-	-	-	-	-	-

**Abbildung 8: Benötigte Signale (Siehe ISO 15118 Teil 2), Seite 117)**

Auf den Seiten 142–149 wird das Kommunikations-Timing abgehandelt. Hier werden die zeitlichen Abstände jeder Nachricht von ihrer vorherigen Nachricht definiert. Innerhalb von diesen Zeiten müssen die Nachrichten vollständig übermittelt sein. Falls dies nicht der Fall ist, wird die Kommunikation beendet.

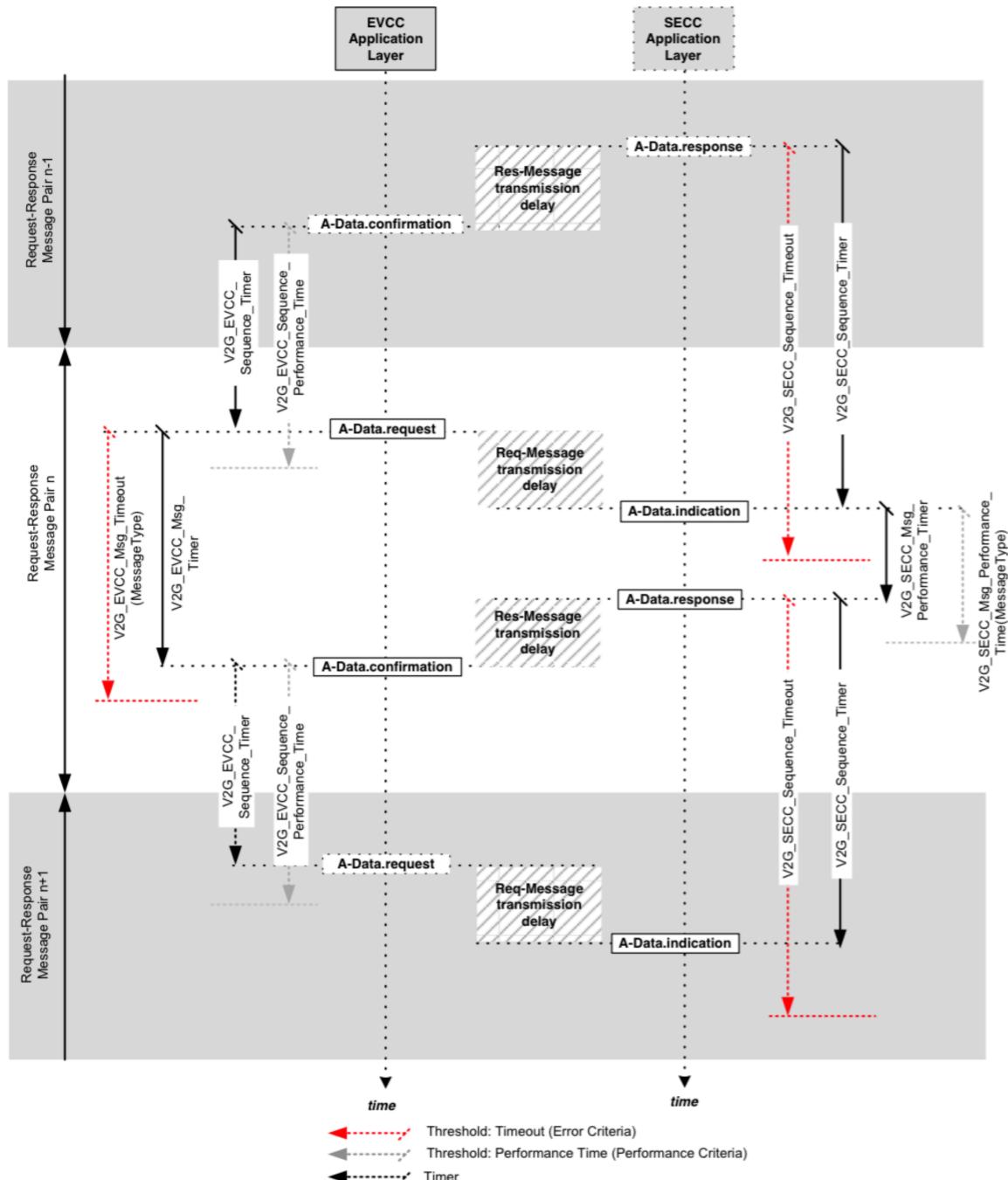


Figure 93 — Message sequence and session timing

Abbildung 9: Zeitliches Verhalten von Botschaften (Siehe (ISO 15118 Teil 2), Seite 145)

#### 4.2.4 Response-Nachrichten

Auf den Seiten 149-155 ist die „table 106“ abgebildet, welche für jede Botschaft die Antwortmöglichkeiten des Response-Codes vorgibt. Somit ist klar definiert, welche Fehlermeldungen angezeigt werden können.

**ISO/IEC DIS 15118-2**

**Table 106 — Overview on application of ResponseCodes**

Response Code (Enum)	V2G message type																										
	supportedAppProtocolReq	supportedAppProtocolRes	SessionSetupReq	SessionSetupRes	ServiceDiscoveryReq	ServiceDiscoveryRes	ServiceDetailReq	ServiceDetailRes	ServiceandPaymentSelectiReq	ServiceandPaymentsSelectionRes	PaymentDetailsReq	PaymentDetailsRes	ContractAuthenticationReq	ContractAuthenticationRes	ChargeParameterDiscoveryReq	ChargeParameterDiscoveryRes	PowerDeliveryReq	PowerDeliveryRes	ChargingStatusReq	ChargingStatusRes	MeteringReceiptReq	MeteringReceiptRes	CertificateupdateReq	CertificateupdateRes	CertificateInstallationReq	CertificateInstallationRes	SessionStopReq
OK	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
OK_CertificateExpiresSoon																											
OK_NewSessionEstablished		x																									
OK_OldSessionJoined		x																									
FAILED	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
FAILED_SequenceError	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
FAILED_SignatureError	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
FAILED_UnknownSession			x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		
FAILED_ServiceIDInvalid				x																							
FAILED_PaymentSelectionInvalid						x																					
FAILED_ServiceSelectionInvalid							x																				
FAILED_CertificateExpired								x												x	x						
FAILED_NoCertificateAvailable									x												x						
FAILED_CertChainError										x											x						
FAILED_ContractCanceled											x										x						
FAILED_ChallengeInvalid											x																
FAILED_WrongEnergyTransferType												x															
FAILED_WrongChargeParameter												x															
FAILED_ChargingProfileInvalid													x														
FAILED_TariffSelectionInvalid													x														
FAILED_PowerDeliveryNotApplied													x														
FAILED_MeteringSignatureNotValid														x													

Abbildung 10: Response Codes (Siehe (ISO 15118 Teil 2), Seite 153)

#### 4.2.5 Zustandsautomaten

Ab Seite 155 werden die Zustandsautomaten beschrieben. In den folgenden beiden Darstellungen (Abbildung 11 & Abbildung 12) sind die Zustandsautomaten einmal für den SECC und für den EVCC zu sehen.

Bei der Darstellung des SECC als Zustandsautomat sind die Zustände jeweils das Warten auf die Anforderungsnachricht des EVCC. Die Pfeile um in den nächsten Zustand zu gelangen sind die Response-Nachrichten des SECC.

Beim EVCC sind die Zustände jeweils das Warten auf die Response-Nachrichten vom SECC. Die Pfeile stellen das Senden der Request-Nachrichten dar.

Welcher Pfeil von einem Zustand gewählt werden muss und was diese aussagen, ist in der ISO auf den Seiten 155–170 beschrieben.

## Automotive Systems Engineering

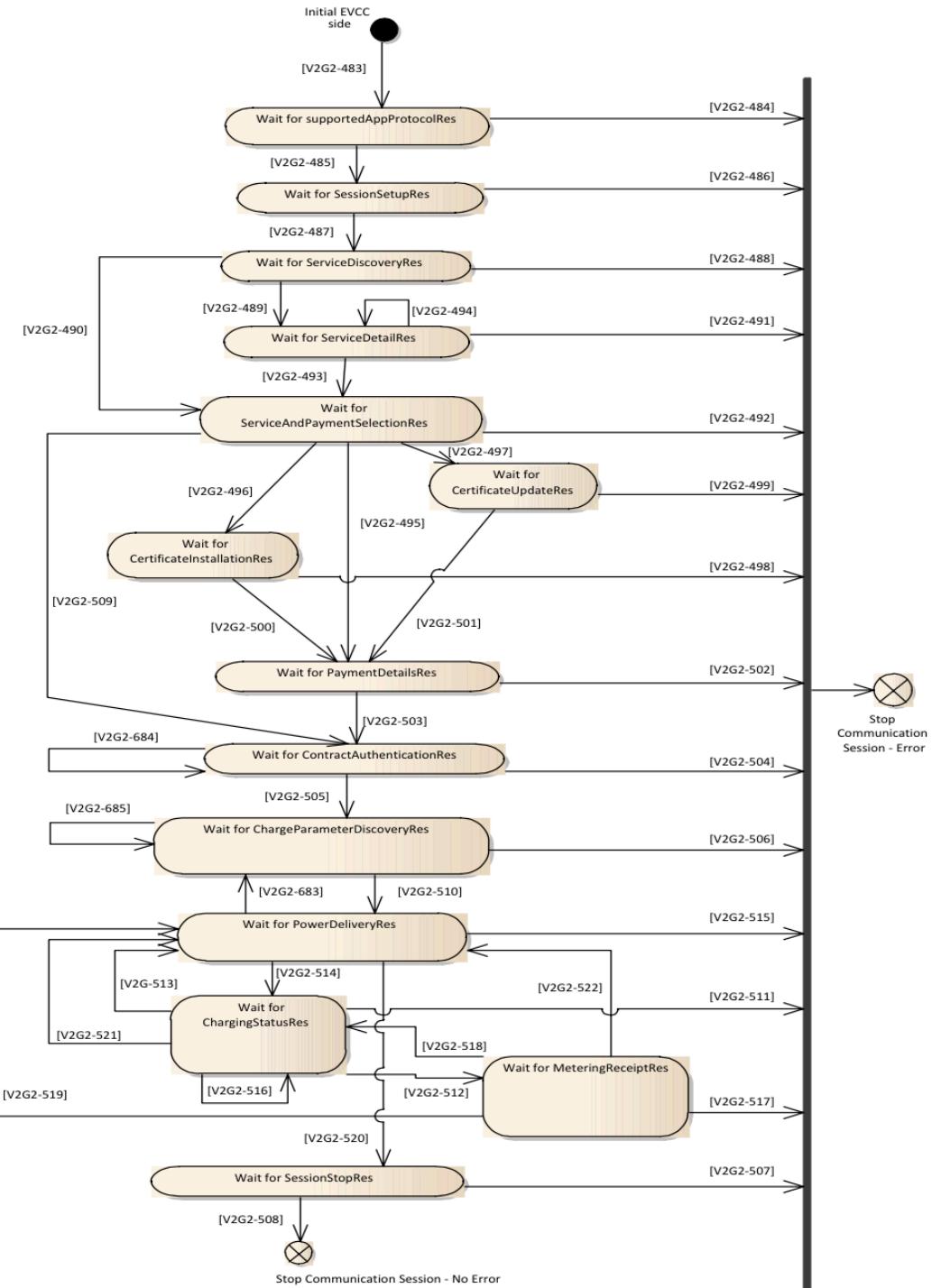


Figure 95 — EVCC Communication states for AC V2G messaging

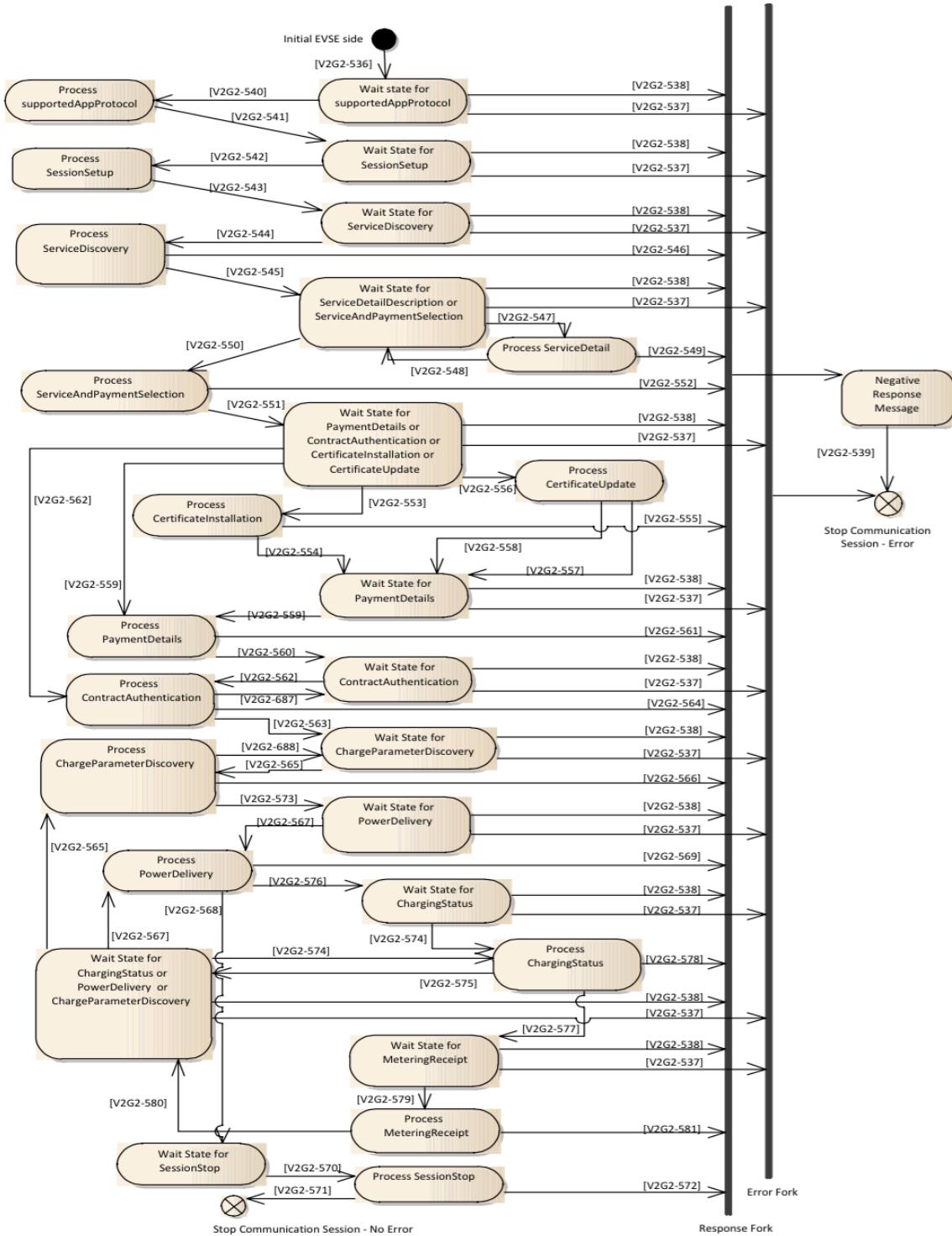


Figure 97 — SECC Communication states for AC V2G messaging

Abbildung 12: Zustandsautomat SECC (Siehe (ISO 15118 Teil 2), Seite 167)

#### 4.2.6 Ablauf Diagramm der Kommunikation

Auf den Seiten 171 bis 179 ist ein Sequenz-Diagramm (Abbildung 13 & Abbildung 14) eines Kommunikationsablaufs ohne Fehler abgebildet. Dies jeweils für AC und DC, jeweils PnC und EIM.

### 8.9 Request-Response Message Sequence Examples

#### 8.9.1 AC

##### 8.9.1.1 EIM

Figure 99 depicts an example for a Request-Response Message Sequence for the EIM identification mode without any errors including optional messages.

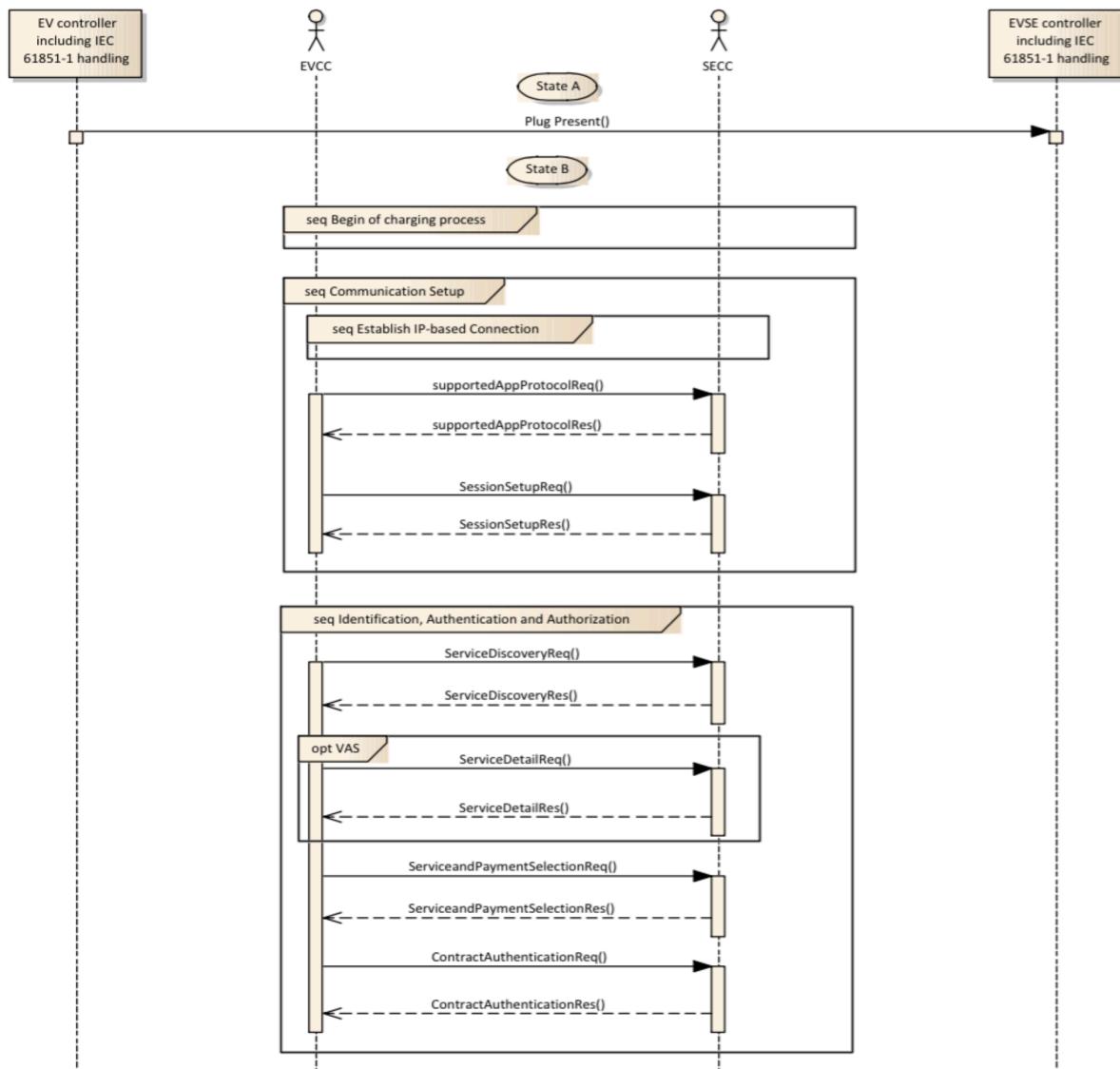


Figure 99 — Overview AC Request-Response Message Sequence EIM identification mode  
(1 of 2)

Abbildung 13: Ablaufdiagramm AC,EIM (Teil 1), (Siehe ISO 15118 Teil 2), Seite 171

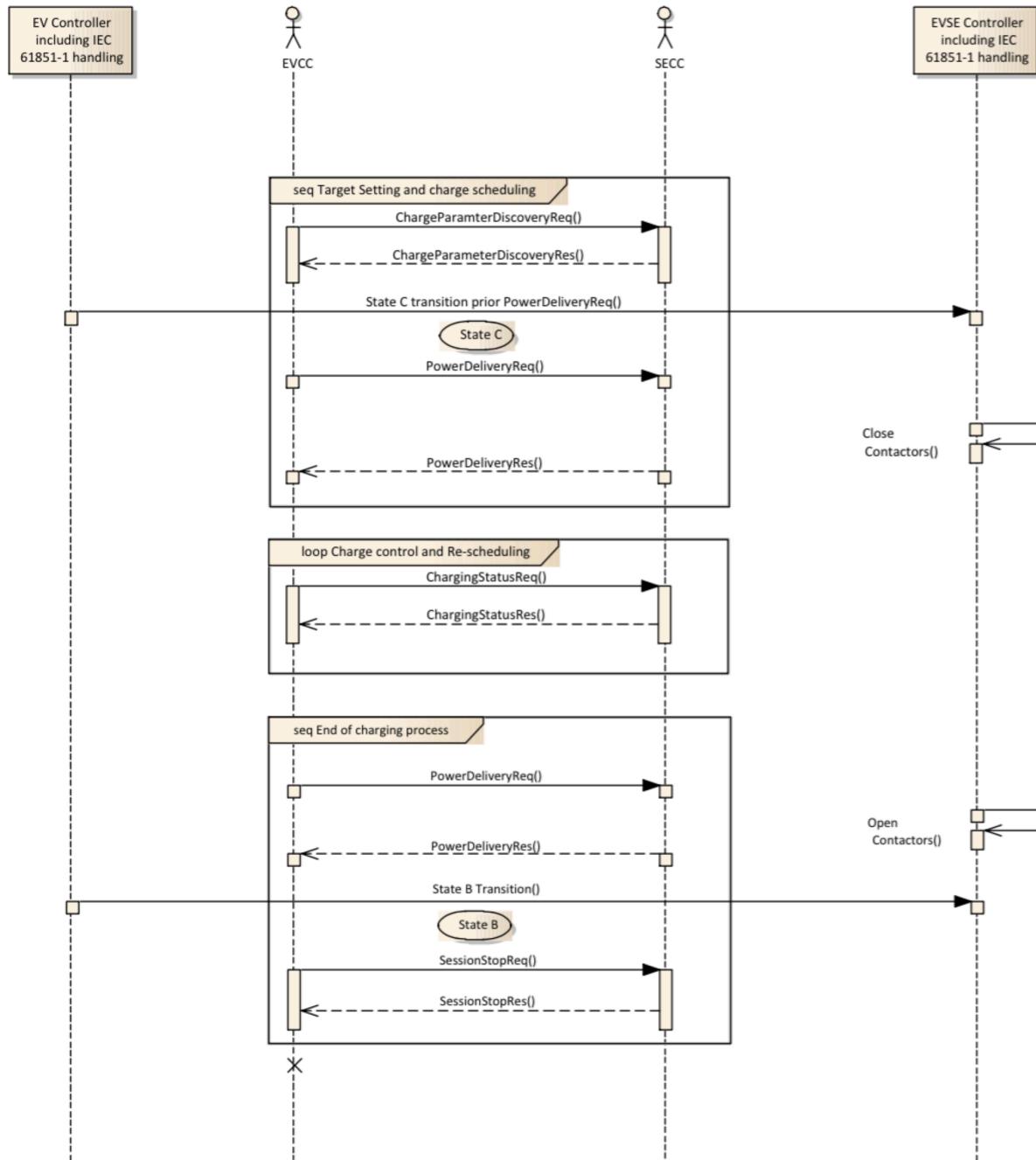


Figure 99— Overview AC Request-Response Message Sequence EIM identification mode  
(2 of 2)

Abbildung 14: Ablaufdiagramm AC,EIM (Teil 2), (Siehe ISO 15118 Teil 2), Seite 172)

#### 4.2.7 Annex A

Hier wird aufgezeigt welcher Block welche Nachrichten beinhaltet.

Es wird aufgezeigt welches Message Set welche Use-Case Elemente beinhaltet.

**Table A. 1— Message Set(s) and covered use case elements**

V2G Message		Message Sets							
Name	Parameter Level	AC Charging EIM	DC Charging EIM	AC Charging PnC	DC Charging PnC	Option: Certificate Update	Option: Certificate Update	Option: MeteringReceipt	Option: VAS
supportedAppProtocolReq		B1	B1	B1	B1	-	-	-	-
	ProtocolNamespace	B1	B1	B1	B1	-	-	-	-
	VersionNumberMajor	B1	B1	B1	B1	-	-	-	-
	VersionNumberMinor	B1	B1	B1	B1	-	-	-	-
	SchemalD	B1	B1	B1	B1	-	-	-	-
	Priority	B1	B1	B1	B1	-	-	-	-
supportedAppProtocolRes		B1	B1	B1	B1	-	-	-	-
	ResponseCode	B1	B1	B1	B1	-	-	-	-
	SchemalD	B1	B1	B1	B1	-	-	-	-
SessionSetupReq		B1	B1	B1	B1	-	-	-	-
	EVCCID	B1	B1	B1	B1	-	-	-	-
SessionSetupRes		B1	B1	B1	B1	-	-	-	-
	ResponseCode	B1	B1	B1	B1	-	-	-	-
	EVSEID	B1	B1	B1	B1	-	-	-	-
	DatetimeNow	-	-	B1	B1	-	-	-	-
ServiceDiscoveryReq		D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-
	ServiceScope	D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-
	ServiceCategory	D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-
ServiceDiscoveryRes		D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-
	ResponseCode	D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-
	PaymentOption	D3, D4	D3, D4	D1, D2	D1, D2	-	-	-	-

Abbildung 15: Nachrichten und Use cases (Siehe (ISO 15118 Teil 2), Seite 180)

#### 4.2.8 Weitere Anhänge

Die Anhänge Annex B bis Annex K sind nicht mehr Bestandteil dieser Dokumentation aber könnten noch weitere zusätzliche Informationen beinhalten.

## 5 Strukturübersicht der Botschaften

### 5.1 Aufbau des Dokuments

Ab Seite 45 bis Seite 114 der ISO (ISO 15118 Teil 2) werden die einzelnen Nachrichten aufgeschlüsselt. Diese Nachrichten haben wir genauer betrachtet. Hierzu haben wir die Beschreibungen ins Deutsche übersetzt und die darin enthaltenen Daten genauer erklärt.

#### Erklärung automatisch erstelltes Word-Dokument:

Als erstes werden die Botschaften erklärt, diese sind im Baum des UML-Chart (Abbildung 16) der oberste Kasten. In der Erklärung wird kurz erläutert, was in der Botschaft passiert. Darunter in der zweiten Ebene werden die komplexen Datentypen (z.B. Word-Dokument, 1.1 GenChallenge) die auch kleinere Datentypen enthalten können, erklärt. Die Datentypen enthalten letztendlich die eigentlich zu übermittelnden Signale. Es können auch direkt in der Botschaft Signale enthalten sein, die keinem Datentyp unterliegen (z.B. Word-Dokument, 1.2 Id).

Die in der ISO vorhandenen Erklärungen der oben genannten Datentypen und Signale wurden herausgesucht um die Variablen zu erklären. Diese Erklärung soll bei der späteren Eingabe der Variablen helfen.

In der folgenden Darstellung (Abbildung 16) ist der komplette Strukturbaum der Botschaften zu sehen, welche in der ISO 15118 verwendet werden.



Abbildung 16: UML- Chart gesamt (Mindjet)

In folgender Abbildung ist nur eine Botschaft zu sehen, welche im Folgenden kurz erklärt wird.

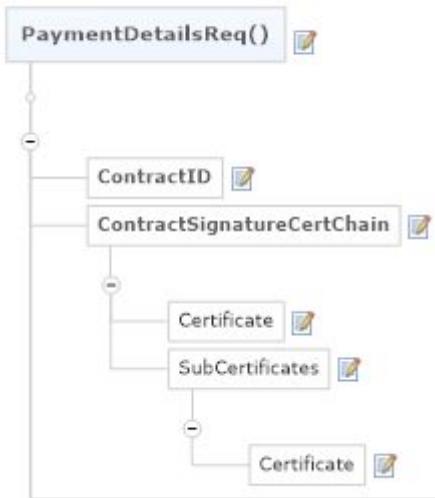


Abbildung 17: Beispiel- Botschaft aus UML- Chart (Mindjet)

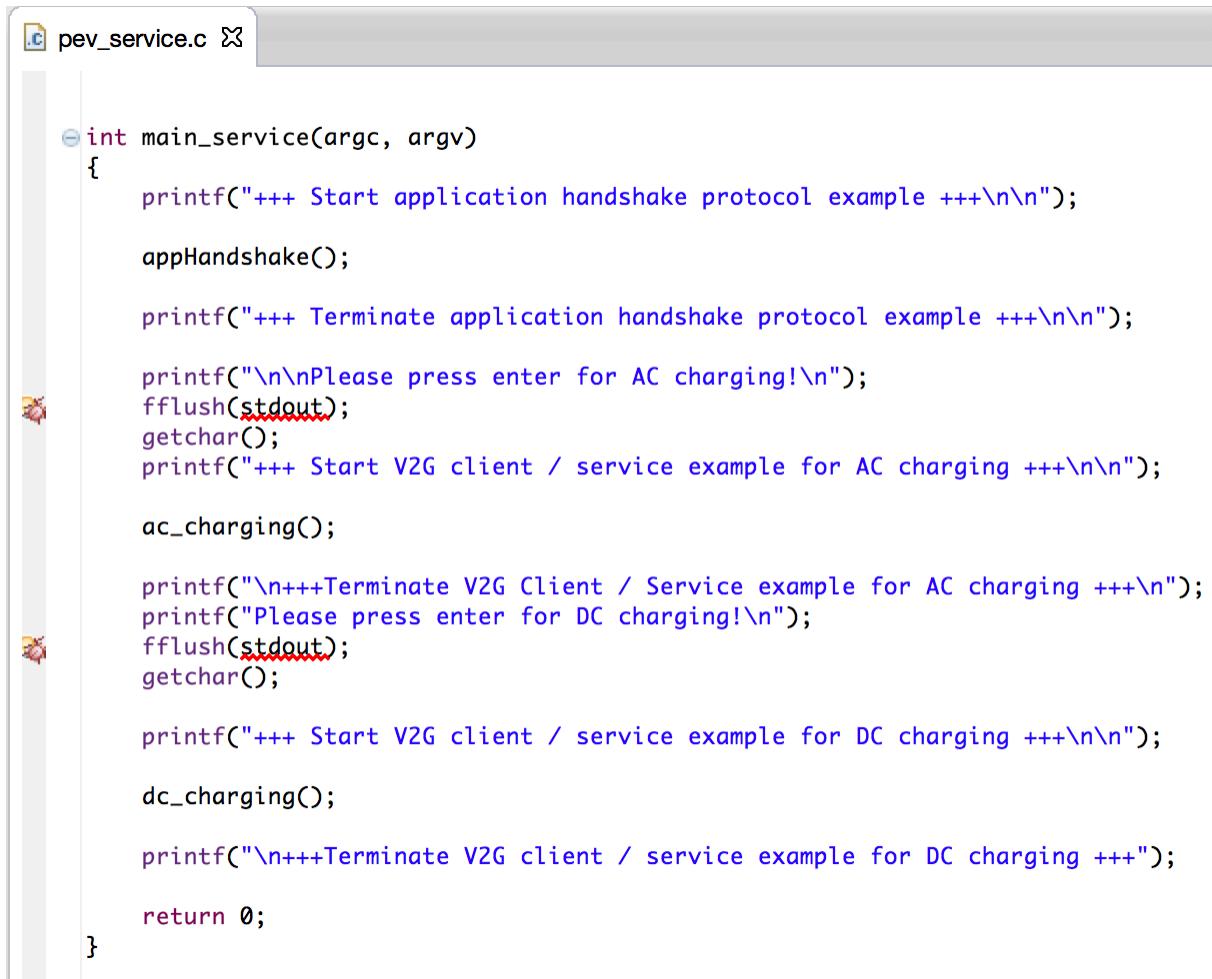
#### PaymentDetailsReq():

In der Botschaft „PaymentDetailsReq()“ bietet der EVCC die Zahlungsdetails an, falls die ausgewählte Zahlungsart "Contract" ist. Mit dem "signature certificate chain" und der "Contract id" fordert der EVCC den SECCD auf eine "challenge" zu senden. Hierbei ist die ContractID eine Nummer die den Ladevertrag identifiziert. Das Format hierzu ist in der DIN 91286 hinterlegt. In jeder Beschreibung der einzelnen Variablen wird immer der Datentyp angegeben, hier bei „ContractID“ ist dies ein String mit einer maximalen Länge von 24 Zeichen. Die „ContractSignatureCertChain“ beinhaltet Zertifikate und optional auch Sub-Zertifikate welche die Kommunikation in diesem Kommunikationsschritt beschreiben.

## 6 Programmbeispiel

In diesem Abschnitt wird auf das vorhandene Beispielprogramm innerhalb des Kommunikationsstack eingegangen. In der Abbildung 18 ist die main\_service Funktion, welche durch die main.c aufgerufen wird, zu finden. Innerhalb dieser

werden drei weitere Funktionen aufgerufen. Die appHandshake, die ac\_charging und die dc\_chargin. Wie die Namen schon vermuten lassen, wird in der ersten Funktion der Handshake zwischen Fahrzeug und Ladesäule simuliert und in den zwei weiteren das Gleichstrom- und das Wechselstromladen. Der Rest dient nur als Anzeigetext in der Konsole und als Erklärung des Vorgehens.



```

pev_service.c

int main_service(argc, argv)
{
    printf("+++ Start application handshake protocol example +++\n\n");
    appHandshake();

    printf("+++ Terminate application handshake protocol example +++\n\n");
    printf("\n\nPlease press enter for AC charging!\n");
    fflush(stdout);
    getchar();
    printf("+++ Start V2G client / service example for AC charging +++\n\n");
    ac_charging();

    printf("\n+++Terminate V2G Client / Service example for AC charging +++\n");
    printf("Please press enter for DC charging!\n");
    fflush(stdout);
    getchar();

    printf("+++ Start V2G client / service example for DC charging +++\n\n");
    dc_charging();

    printf("\n+++Terminate V2G client / service example for DC charging +++");
    return 0;
}
  
```

Abbildung 18: Beispielprogramm Bild 1

Die Abbildung 20, Fehler! Verweisquelle konnte nicht gefunden werden. und Abbildung 21 zeigen die appHandshake Funktion. In dieser werden die Parameter befüllt und der Handshake vorgenommen.

```

pev_service.c ✘
static int appHandshake()
{
    static uint8_t byte_array[MAX_BYTE_SIZE]; /* define MAX_BYTE_SIZE before*/
    static uint32_t string_array[MAX_STRING_SIZE]; /* define MAX_STRING_SIZE before*/

    /* define in and out byte stream */
    uint8_t inStream[MAX_STREAM_SIZE]; /* define MAX_STREAM_SIZE before */
    uint8_t outStream[MAX_STREAM_SIZE]; /* define MAX_STREAM_SIZE before */

    /* BINARY memory setup */
    exi_bytes_t bytes = { MAX_BYTE_SIZE, byte_array, 0 };

    /* STRING memory setup */
    exi_string_ucs_t string = { MAX_STRING_SIZE, string_array, 0 };

    struct EXIDatabinder appHandService;
    struct EXIDocumentType_appHand exiDoc;
    struct SupportedAppProtocolReq handshake;
    struct SupportedAppProtocolRes resultHandshake;
    uint16_t length, payloadLength;

    /* init the app handshake serializer.
     * Important: also provide the offset of the V2GTP header length */
    init_appHandSerializer(&appHandService,bytes,string,MAX_STREAM_SIZE, V2GTP_HEADER_LENGTH);

    init_EXIDocumentType_appHand(&exiDoc);

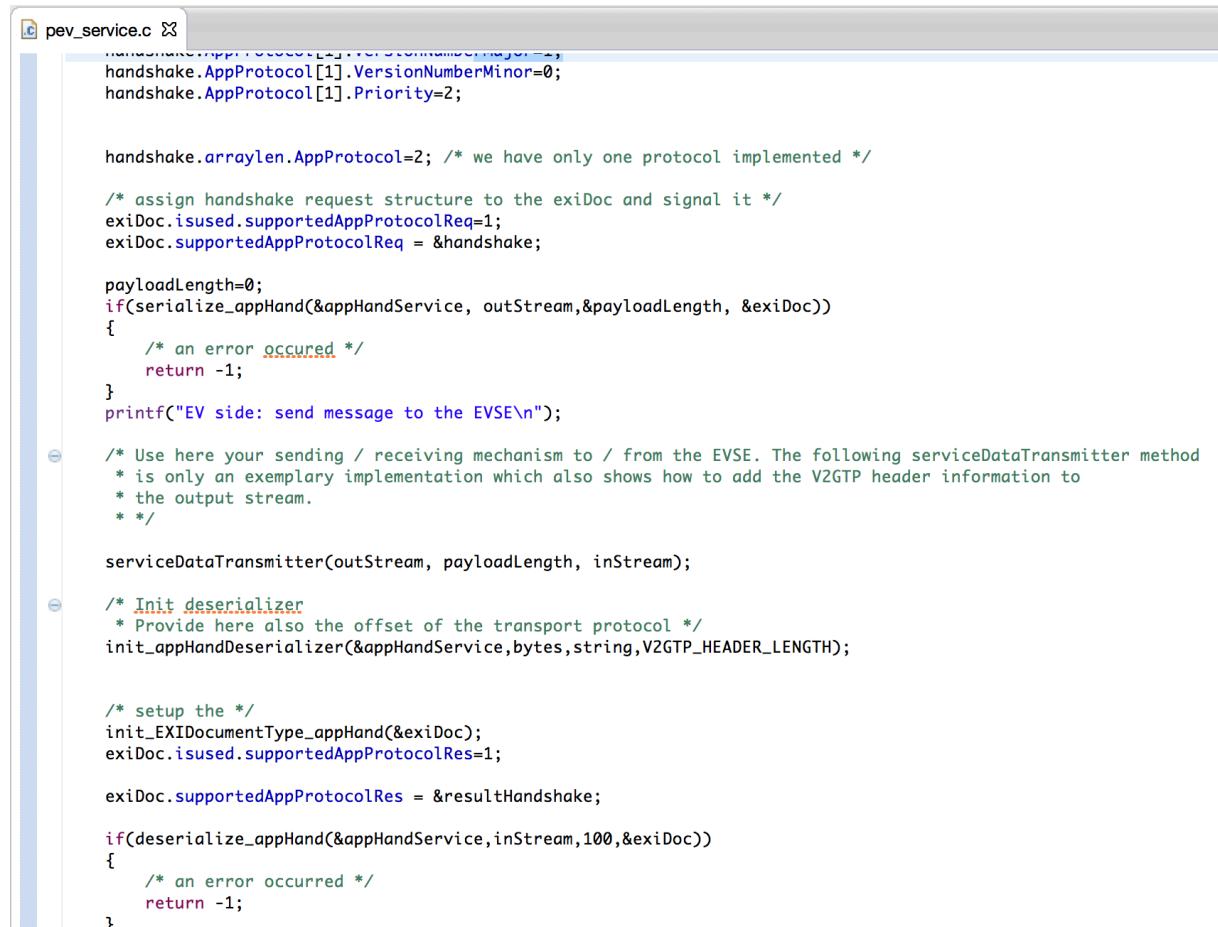
    printf("EV side: setup data for the supported application handshake request message\n");

    /* set up ISO/IEC 15118 Version 1.0 information */
    length = writeStringToEXIString("urn:iso:15118:2:2010:MsgDef", handshake.AppProtocol[0].ProtocolNamespace.data);
    handshake.AppProtocol[0].ProtocolNamespace.arraylen.data = length; /* length of the string */
    handshake.AppProtocol[0].SchemaID=1;
    handshake.AppProtocol[0].VersionNumberMajor=1;
    handshake.AppProtocol[0].VersionNumberMinor=0;
    handshake.AppProtocol[0].Priority=1;

    length = writeStringToEXIString("urn:dim:70121:2012:MsgDef", handshake.AppProtocol[1].ProtocolNamespace.data);
    handshake.AppProtocol[1].ProtocolNamespace.arraylen.data = length; /* length of the string */
    handshake.AppProtocol[1].SchemaID=2;
    handshake.AppProtocol[1].VersionNumberMajor=1;

```

Abbildung 19: Beispielprogramm Bild 2



```

pev_service.c

handshake.AppProtocol[1].VersionNumberMajor=1,
handshake.AppProtocol[1].VersionNumberMinor=0;
handshake.AppProtocol[1].Priority=2;

handshake.arraylen.AppProtocol=2; /* we have only one protocol implemented */

/* assign handshake request structure to the exiDoc and signal it */
exiDoc.isused.supportedAppProtocolReq=1;
exiDoc.supportedAppProtocolReq = &handshake;

payloadLength=0;
if(serialize_appHand(&appHandService, outStream,&payloadLength, &exiDoc))
{
    /* an error occurred */
    return -1;
}
printf("EV side: send message to the EVSE\n");

/* Use here your sending / receiving mechanism to / from the EVSE. The following serviceDataTransmitter method
 * is only an exemplary implementation which also shows how to add the V2GTP header information to
 * the output stream.
 */
serviceDataTransmitter(outStream, payloadLength, inStream);

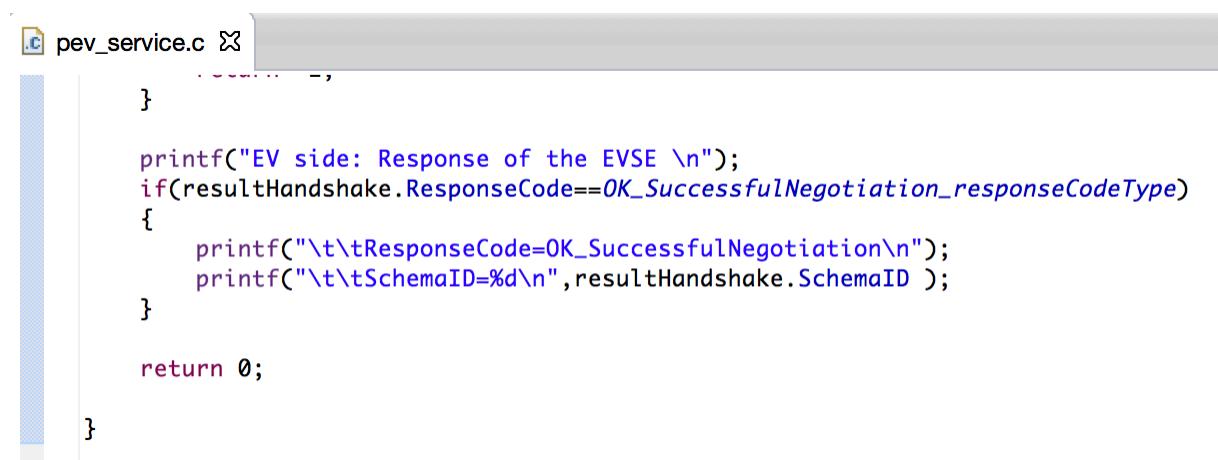
/* Init deserializer
 * Provide here also the offset of the transport protocol */
init_appHandDeserializer(&appHandService,bytes,string,V2GTP_HEADER_LENGTH);

/* setup the */
init_EXIDocumentType_appHand(&exiDoc);
exiDoc.isused.supportedAppProtocolRes=1;

exiDoc.supportedAppProtocolRes = &resultHandshake;

if(deserialize_appHand(&appHandService,inStream,100,&exiDoc))
{
    /* an error occurred */
    return -1;
}
  
```

Abbildung 20: Beispielprogramm Bild 3



```

        }

printf("EV side: Response of the EVSE \n");
if(resultHandshake.ResponseCode==OK_SuccessfulNegotiation_responseCodeType)
{
    printf("\t\tResponseCode=OK_SuccessfulNegotiation\n");
    printf("\t\tSchemaID=%d\n",resultHandshake.SchemaID );
}

return 0;
}
  
```

Abbildung 21: Beispielprogramm Bild 4

Die Abbildung 22 bis Abbildung 28 zeigen den Code der ac\_charging Funktion. Im ersten Schritt werden alle nötigen Botschaften als Struct erstellt. Nun wird mit der init\_v2gServiceClient der instream und outstream deklariert und mit dem „service“

verbunden. Mit der Funktion `prepare_sessionSetup` werden die zuvor eingegebenen `sessionSetup` Daten in den „`service`“ struct geschrieben. Dieser wird dann durch die Funktion `serviceDataTransmitter` übertragen. Die Funktion muss zum Teil noch programmiert werden sobald eine tatsächliche Übertragung stattfinden soll. Im Wesentlichen werden die Daten immer befüllt, anschließend in den „`service`“ geschrieben und dieser dann Übertragen. Im Folgenden wird die Kommunikation in ein paar Funktionen beschrieben. Diese sind die Hauptfunktionen der Rest dient dazu die Werte in die Structs zu schreiben und Informationen auf der Konsole auszugeben.

`ac_charging`:

```
init_v2gServiceClient  
prepare_sessionSetup → schreibt Daten in den „service“  
serviceDataTransmitter → sendet den „service“ an die Ladesäule  
prepare_serviceDiscovery  
serviceDataTransmitter  
prepare_serviceDetail  
serviceDataTransmitter  
prepare_paymentServiceSelektion  
serviceDataTransmitter
```

...

Nach diesem Schema läuft das komplette Beispiel ab.

pev\_service.c

```

static int ac_charging()
{
    static uint8_t byte_array[MAX_BYTE_SIZE]; /* define MAX_BYTE_SIZE before*/
    static uint32_t string_array[MAX_STRING_SIZE]; /* define MAX_STRING_SIZE before*/

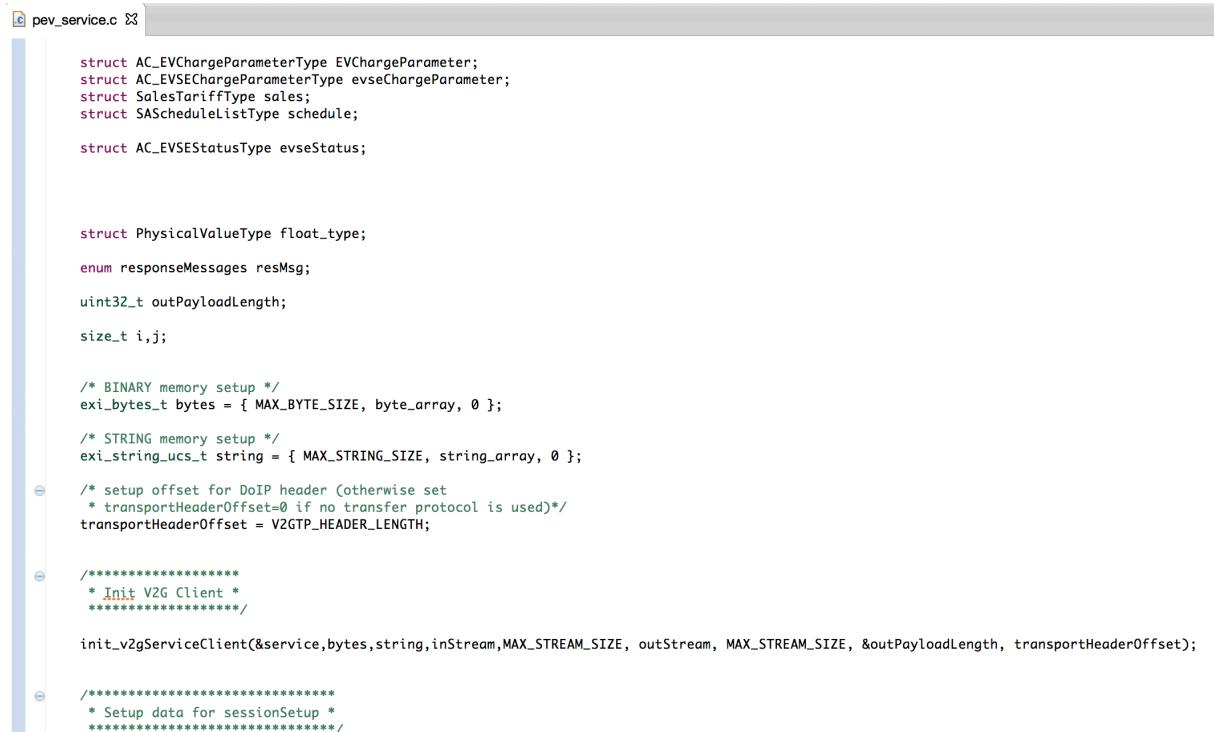
    /* define in and out byte stream */
    uint8_t inStream[MAX_STREAM_SIZE]; /* define MAX_STREAM_SIZE before */
    uint8_t outStream[MAX_STREAM_SIZE]; /* define MAX_STREAM_SIZE before */

    /* define offset variable for transport header data */
    uint16_t transportHeaderOffset;

    /* service data structure for AC*/
    struct EXIService service;
    struct MessageHeaderType v2gHeader;
    struct SessionSetupReqType sessionSetup;
    struct SessionSetupResType resultSessionSetup;
    struct ServiceDiscoveryReqType serviceDiscovery;
    struct ServiceDiscoveryResType resultServiceDiscovery;
    struct ServiceDetailReqType serviceDetail;
    struct ServiceDetailResType resultServiceDetail;
    struct PaymentServiceSelectionReqType servicePayment;
    struct PaymentServiceSelectionResType resultServicePayment;
    struct PaymentDetailsReqType paymentDetails;
    struct PaymentDetailsResType resultPaymentDetails;
    struct AuthorizationReqType contractAuthentication;
    struct AuthorizationResType resultContractAuthentication;
    struct ChargeParameterDiscoveryReqType powerDiscovery;
    struct ChargeParameterDiscoveryResType resultPowerDiscovery;
    struct PowerDeliveryReqType powerDelivery;
    struct PowerDeliveryResType resultPowerDelivery;
    struct ChargingStatusResType resultChargingStatus;
    struct MeteringReceiptReqType meteringReceipt;
    struct MeteringReceiptResType resultMeteringReceipt;
    struct SessionStopReqType sessionStop;
    struct SessionStopResType resultSessionStop;
}

```

Abbildung 22: Beispielprogramm Bild 5



```

pev_service.c

struct AC_EVChargeParameterType EVChargeParameter;
struct AC_EVSEChargeParameterType evseChargeParameter;
struct SalesTariffType sales;
struct SAScheduleListType schedule;

struct AC_EVSEStatusType evseStatus;

struct PhysicalValueType float_type;
enum responseMessages resMsg;
uint32_t outPayloadLength;
size_t i,j;

/* BINARY memory setup */
exi_bytes_t bytes = { MAX_BYTE_SIZE, byte_array, 0 };

/* STRING memory setup */
exi_string_ucs_t string = { MAX_STRING_SIZE, string_array, 0 };

transportHeaderOffset = V2GTP_HEADER_LENGTH;

*****
* Init V2G Client *
*****

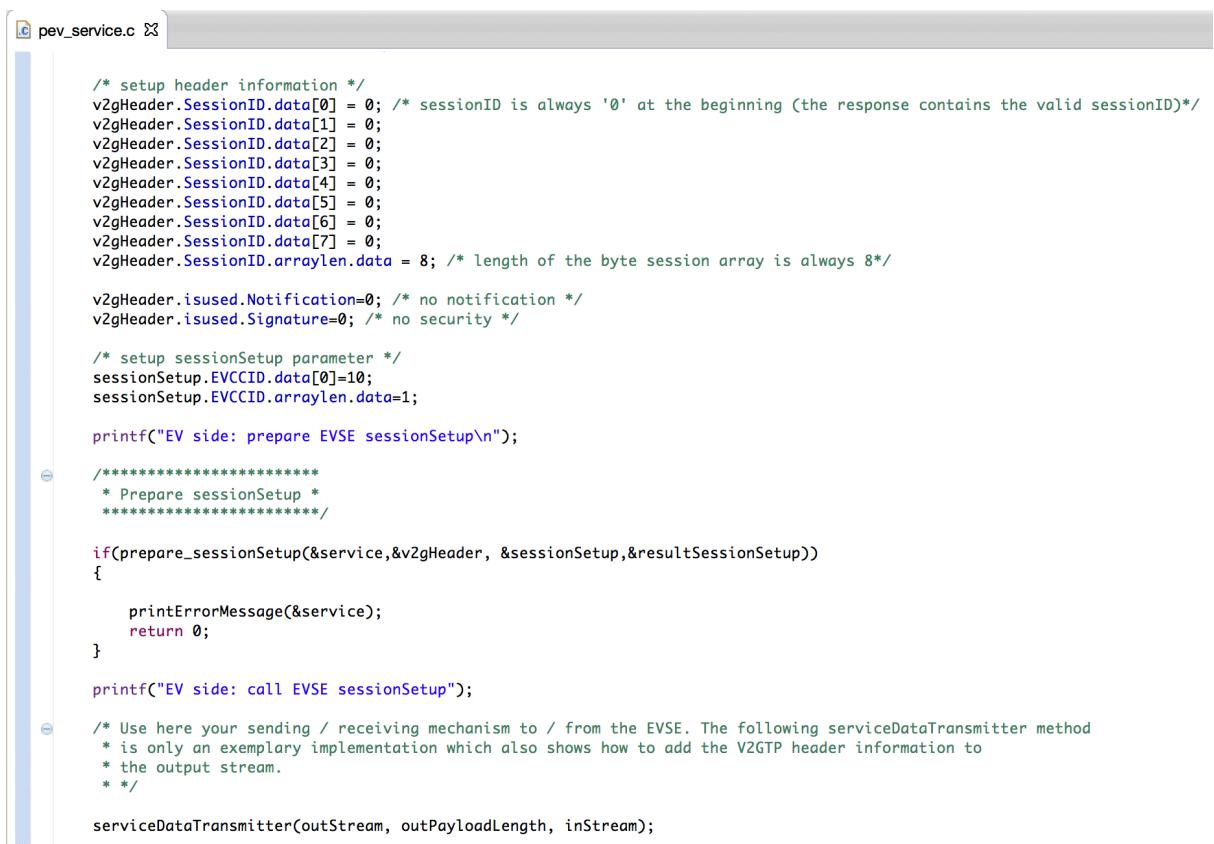
init_v2gServiceClient(&service,bytes,string,inStream,MAX_STREAM_SIZE, outStream, MAX_STREAM_SIZE, &outPayloadLength, transportHeaderOffset);

*****
* Setup data for sessionSetup *
*****

printf("EV side: call EVSE sessionSetup");

```

Abbildung 23: Beispielprogramm Bild 6



```

pev_service.c

/* setup header information */
v2gHeader.SessionID.data[0] = 0; /* sessionID is always '0' at the beginning (the response contains the valid sessionID)*/
v2gHeader.SessionID.data[1] = 0;
v2gHeader.SessionID.data[2] = 0;
v2gHeader.SessionID.data[3] = 0;
v2gHeader.SessionID.data[4] = 0;
v2gHeader.SessionID.data[5] = 0;
v2gHeader.SessionID.data[6] = 0;
v2gHeader.SessionID.data[7] = 0;
v2gHeader.SessionID.arraylen.data = 8; /* length of the byte session array is always 8*/

v2gHeader.isused.Notification=0; /* no notification */
v2gHeader.isused.Signature=0; /* no security */

/* setup sessionSetup parameter */
sessionSetup.EVCCID.data[0]=10;
sessionSetup.EVCCID.arraylen.data=1;

printf("EV side: prepare EVSE sessionSetup\n");

*****
* Prepare sessionSetup *
*****

if(prepare_sessionSetup(&service,&v2gHeader, &sessionSetup,&resultSessionSetup))
{
    printErrorMessage(&service);
    return 0;
}

printf("EV side: call EVSE sessionSetup");

/* Use here your sending / receiving mechanism to / from the EVSE. The following serviceDataTransmitter method
 * is only an exemplary implementation which also shows how to add the V2GTP header information to
 * the output stream.
 * */
serviceDataTransmitter(outStream, outPayloadLength, inStream);

```

Abbildung 24: Beispielprogramm Bild 7

```

pev_service.c ✘

/* this methods deserialize the response EXI stream */
if( determineResponseMessage(&service, &resMsg))
{
    printErrorMessage(&service);
    return 0;
}

/* check, if this is the sessionSetup response message */
if(resMsg==SESSIONSETUPRES)
{
    /* show result of the answer message of EVSE sessionSetup */
    printf("EV side: received response message from EVSE\n");
    printf("\tHeader SessionID=%d",v2gHeader.SessionID.data,v2gHeader.SessionID.arraylen.data );
    printf("\tResponseCode=%d\n",resultSessionSetup.ResponseCode);
    printf("\tEVSEID=%d\n", resultSessionSetup.EVSEID.data[1]);
    printf("\tEVSETimeStamp=%d\n",resultSessionSetup.EVSETimeStamp);

}

*****
* Setup data for serviceDiscovery *
***** 

serviceDiscovery.isUsed.ServiceCategory=1;
serviceDiscovery.ServiceCategory = Internet_serviceCategoryType;
serviceDiscovery.isUsed.ServiceScope=0;

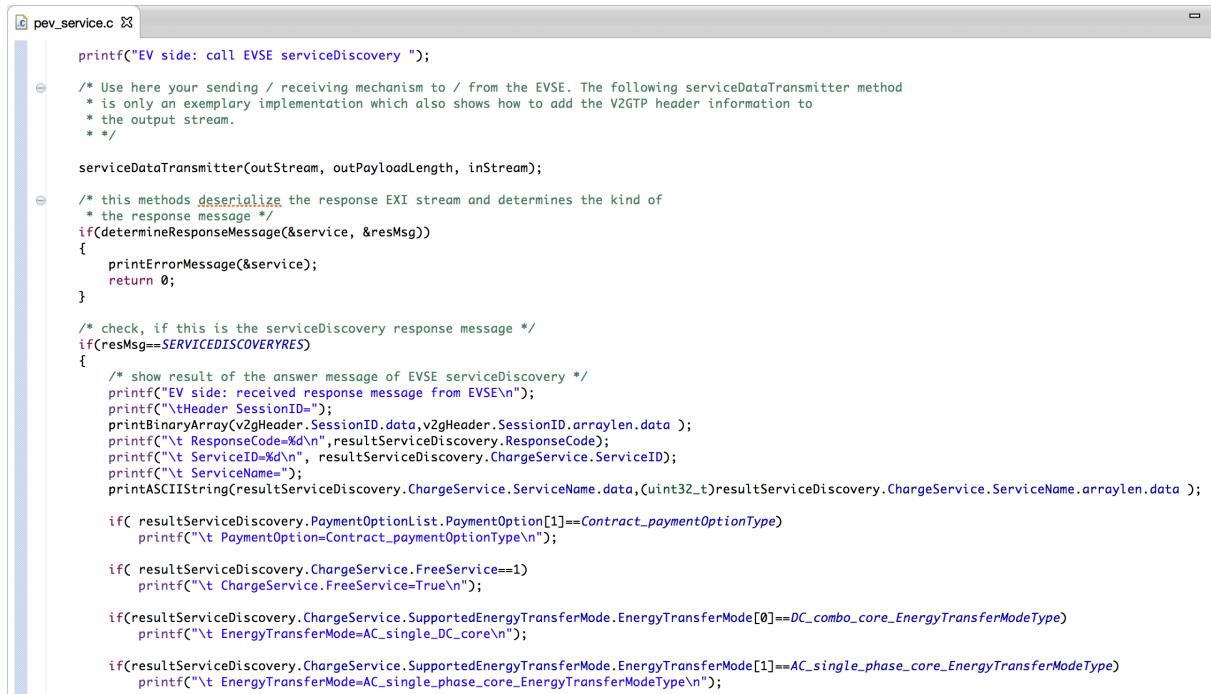
printf("\n\nEV side: prepare EVSE serviceDiscovery\n");

*****
* Prepare serviceDiscovery *
***** 

prepare_serviceDiscovery(&service,&v2gHeader, &serviceDiscovery,&resultServiceDiscovery);

```

Abbildung 25: Beispielprogramm Bild 8



```

printf("EV side: call EVSE serviceDiscovery ");

/* Use here your sending / receiving mechanism to / from the EVSE. The following serviceDataTransmitter method
 * is only an exemplary implementation which also shows how to add the V2GTP header information to
 * the output stream.
 */

serviceDataTransmitter(outStream, outPayloadLength, inStream);

/* this methods deserialize the response EXI stream and determines the kind of
 * the response message */
if(determineResponseMessage(&service, &resMsg))
{
    printErrorMessage(&service);
    return 0;
}

/* check, if this is the serviceDiscovery response message */
if(resMsg==SERVICEDISSCOVERYRES)
{
    /* show result of the answer message of EVSE serviceDiscovery */
    printf("EV side: received response message from EVSE\n");
    printf("\tHeader.SessionID.data,v2gHeader.SessionID.arraylen.data ");
    printf("\t\tResponseCode=%d\n", resultServiceDiscovery.ResponseCode);
    printf("\t\tServiceID=%d\n", resultServiceDiscovery.ChargeService.ServiceID);
    printf("\t\tServiceName=");
    printASCIIString(resultServiceDiscovery.ChargeService.ServiceName.data,(uint32_t)resultServiceDiscovery.ChargeService.ServiceName.arraylen.data );

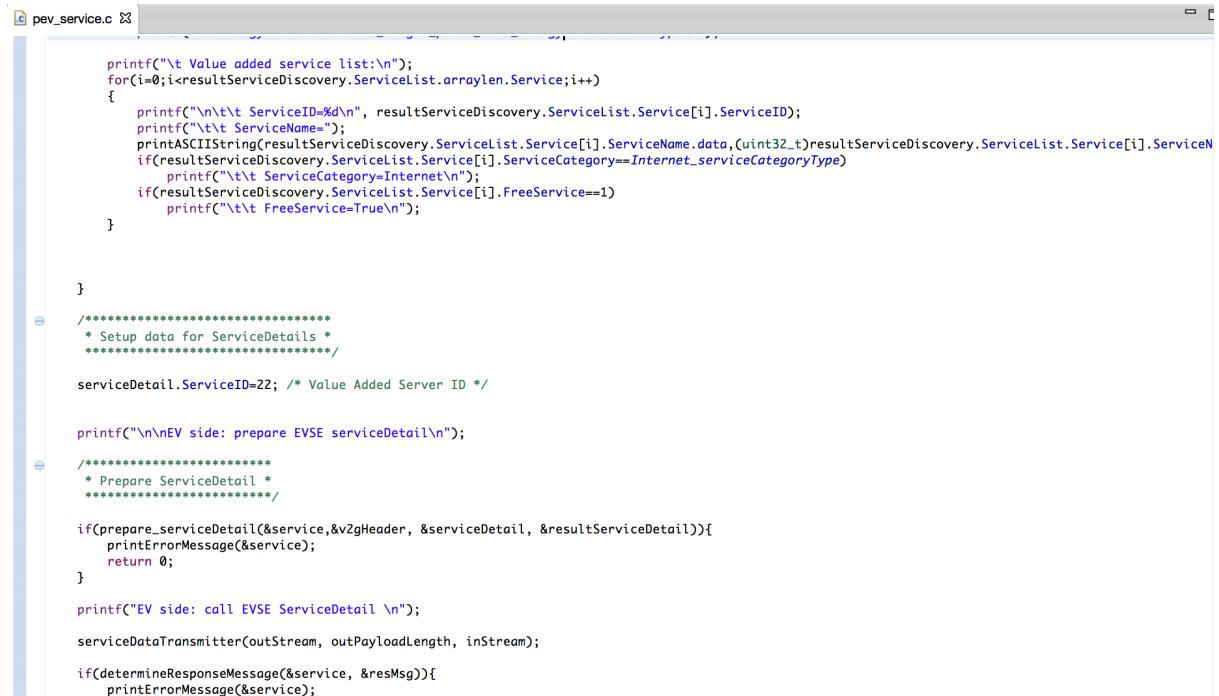
    if(resultServiceDiscovery.PaymentOptionList.PaymentOption[1]==Contract_paymentOptionType)
        printf("\t\tPaymentOption=Contract_paymentOptionType\n");

    if(resultServiceDiscovery.ChargeService.FreeService==1)
        printf("\t\tChargeService.FreeService=True\n");

    if(resultServiceDiscovery.ChargeService.SupportedEnergyTransferMode.EnergyTransferMode[0]==DC_combo_core_EnergyTransferModeType)
        printf("\t\tEnergyTransferMode=AC_single_DC_core\n");

    if(resultServiceDiscovery.ChargeService.SupportedEnergyTransferMode.EnergyTransferMode[1]==AC_single_phase_core_EnergyTransferModeType)
        printf("\t\tEnergyTransferMode=AC_single_phase_core_EnergyTransferModeType\n");
}

```

**Abbildung 26: Beispielprogramm Bild 9**


```

printf("\t Value added service list:\n");
for(i=0;i<resultServiceDiscovery.ServiceList.arraylen.Service;i++)
{
    printf("\n\t\t ServiceID=%d\n", resultServiceDiscovery.ServiceList.Service[i].ServiceID);
    printf("\t\t ServiceName=");
    printASCIIString(resultServiceDiscovery.ServiceList.Service[i].ServiceName.data,(uint32_t)resultServiceDiscovery.ServiceList.Service[i].ServiceName.arraylen.data );
    if(resultServiceDiscovery.ServiceList.Service[i].ServiceCategory==Internet_serviceCategoryType)
        printf("\t\t ServiceCategory=Internet\n");
    if(resultServiceDiscovery.ServiceList.Service[i].FreeService==1)
        printf("\t\t FreeService=True\n");
}

}

/*****************
 * Setup data for ServiceDetails *
*****************/
serviceDetail.ServiceID=22; /* Value Added Server ID */

printf("\n\nEV side: prepare EVSE serviceDetail\n");

/* ****
 * Prepare ServiceDetail *
****/

if(prepare_serviceDetail(&service,&v2gHeader, &serviceDetail, &resultServiceDetail)){
    printErrorMessage(&service);
    return 0;
}

printf("EV side: call EVSE ServiceDetail \n");

serviceDataTransmitter(outStream, outPayloadLength, inStream);

if(determineResponseMessage(&service, &resMsg)){
    printErrorMessage(&service);
}

```

**Abbildung 27: Beispielprogramm Bild 10**

```

pev_service.c ①

/*
 * Setup data for ServicePaymentSelection *
 ****
servicePayment.SelectedPaymentOption = ExternalPayment_paymentOptionType;
servicePayment.SelectedServiceList.SelectedService[0].ServiceID=resultServiceDiscovery.ChargeService.ServiceID; /* charge server ID */
servicePayment.SelectedServiceList.SelectedService[0].isused.ParameterSetID=0; /* is not used */
servicePayment.SelectedServiceList.SelectedService[1].ServiceID = resultServiceDiscovery.ServiceList.Service[0].ServiceID;
servicePayment.SelectedServiceList.SelectedService[1].ParameterSetID=resultServiceDetail.ServiceParameterList.ParameterSet[0].ParameterSetID;
servicePayment.SelectedServiceList.SelectedService[1].isused.ParameterSetID=1;
servicePayment.SelectedServiceList.arraylen.SelectedService=2; /* only one service was selected */

printf("\n\nEV side: prepare EVSE servicePaymentSelection\n");

/*
 * Prepare ServicePaymentSelection *
 ****
if(prepare_paymentServiceSelection(&service,&v2gHeader, &servicePayment,&resultServicePayment))
{
    printErrorMessage(&service);
    return 0;
}

printf("EV side: call EVSE ServicePaymentSelection \n");

/* Use here your sending / receiving mechanism to / from the EVSE. The following serviceDataTransmitter method
 * is only an exemplary implementation which also shows how to add the V2GTP header information to
 * the output stream.
 */
serviceDataTransmitter(outStream, outPayloadLength, inStream);

```

**Abbildung 28: Beispielprogramm Bild 11**