D

SELVE

Technik, die bewegt

SELVE XML Spezifikation

Schnittstellenbeschreibung

Revision **2.0.2**

Stand **Juni 2016**

SELVE GmbH & Co.KG ● Werdohler Landstr. 286 ● D – 58513 Lüdenscheid

Inhaltsverzeichnis

Bildverzeichnis

Tabellenverzeichnis

1 Grundlagen

Mittels der vorliegenden „SELVE XML-Spezifikation“ können über das SELVE USB-RF Gateway (SELVE Artikel Nr. 297792) die Produkte mit den SELVE Funktechnologien **commeo** und **iveo** bedient werden. Somit ist es möglich Funktionen der Hausautomation in eigene Anwendungen einfach und schnell zu integrieren.

Zunächst werden in diesem Kapitel die Grundlagen beschrieben und zeigen anhand von Beispielen die Integration, bzw. den Aufbau der Telegrammstruktur. In den folgenden Kapiteln werden anschließend die einzelnen XML Methoden aufgeführt, die das SELVE USB-RF Gateway unterstützt.

Im Allgemeinen umfasst die Kommunikationsschnittstelle folgende Funktionen:

- Grundlegende Kommunikation und Einstellungen des USB-RF Gateways.
- Einlernen und Verwalten von bis zu 64 **commeo** Motore / Aktore.
- Einzel-, und Gruppenbedienung der **commeo** Motore / Aktore.
- Einlernen von bis zu 8 **commeo** Sensoren für Visualisierungen.
- Einlernen von bis zu 63 **commeo** Handsenderkanäle für einfache Bedienungen in Fremdsysteme.
- Einrichten von bis zu 32 vordefinierten **commeo** Gruppen.
- Einrichten von bis zu 8 **commeo** Sensorsimulationen als Schnittstelle zur Nutzung von Sensoren aus Fremdsysteme.
- Firmwareaktualisierung des SELVE USB RF Gateways.
- Simulation von bis zu 64 **iveo** Funkkanäle zur Steuerung von iveo Antrieben.

1.1 Schematischer Aufbau des SELVE USB-RF Gateways

Die folgende Abbildung zeigt den schematischen Aufbau des SELVE USB-RF Gateways.

Im unteren Bereich wird die Schnittstelle der externen Anwendung zum eigentlichen Gateway gezeigt. Diese spricht mit der vorliegenden SELVE XML Spezifikation via USB (serieller COM-Port) mit dem Gateway. Über diese Schnittstelle lassen sich verschiedenen **commeo** und **iveo** Geräte einlernen, steuern und überwachen.

Die Logikschicht des SELVE USB-RF Gateways setzt sich im Folgenden aus fünf grundlegenden Teilbereiche zusammen. Ein für SELVE optimierter XML-Interpreter dient für die reibungslose Kommunikation zwischen der externen Anwendung und dem Gateway.

Die Programmlogik sorgt für die internen Verknüpfungen und Verzweigungen zwischen der XML-Schnittstelle, der internen Logik und den verfügbaren Funktechnologien, die sich zwischen **commeo** und **iveo** aufteilen. Weiterhin verwaltet die Logikschicht die verfügbaren Daten wie Namen, Geräteinformationen, Gateway-Einstellungen, etc. Diese Daten sind im Gateway permanent gespeichert, so dass die Anwendung an dieser Stelle keinerlei zusätzlichen Aufwand tätigen muss.

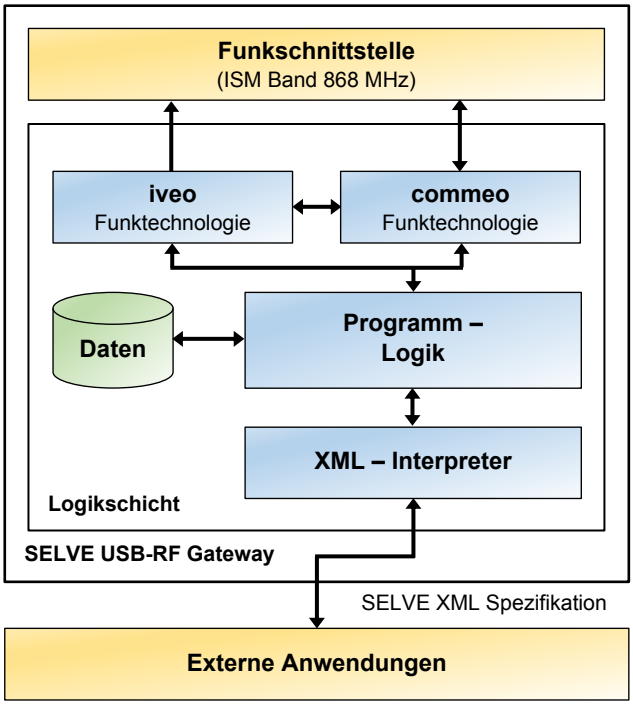

Bild 1. Schematischer Aufbau des SELVE USB-RF Gateways

1.2 XML Schnittstelle

Die Grundlage dieser Schnittstellenbeschreibung (kurz API) liefern die standardisierte XML Sprache (E**x**tensible **M**arkup **L**anguage) und daraus entwickelte XML-RPC Definitionen.

Um den grundsätzlichen Datenaustausch zu beschleunigen und gleichzeitig eine einfache bidirektionale Kommunikation aufbauen zu können wurde in den folgenden Schritten die Kommunikation optimiert. Aufgrund der SELVE spezifischen Optimierungen wird hier von der **SELVE XML Spezifikation** gesprochen. Mit diesen Anpassungen entfällt z.B. ein stetiges Polling des Status durch die Anwendung und minimiert gleichzeitig das notwendige Datenaufkommen, so dass die Schnittstelle auch in kleineren Systemen Platz finden kann.

Für die Schnittstelle sind folgende Eckdaten zu verwenden:

- Schnittstelle: COM Port
- Baudrate: 115.200 Baud
- Daten: 8 Bit
- Parity: keine
- Stop: 1 Bit
- Chip / Treiber: FTDI Chip FT234XD
  Die entsprechenden Treiber können direkt beim Hersteller geladen und genutzt werden (http://www.ftdichip.com/).

Folgende Regeln / Anforderungen sind hierbei zu berücksichtigen:

- Als Zeichen und Datenformat wird die Kodierung **UTF-8** verwendet.
- Die XML-Kommandos und die entsprechenden Antworten sind so ausgelegt, dass eine reibungslose, bidirektionale Kommunikation entstehen kann. Die Zuordnung entsprechender Antworten kann in der Anwendung zu jederzeit gewährleistet werden.
- Die externe Anwendung bekommt auf ein abgesetztes XML Kommando immer eine entsprechende Antwort zurück. Hier wird darauf hingewiesen, dass je nach eingesetztem System Wartezeiten zwischen der XML-Anfrage und der Antwort zu berücksichtigen sind.
- Sendet das SELVE USB-RF Gateway selbstständig ein Ereignis an die Anwendung, so erwartet das Gateway **keinerlei Reaktion** von der Anwendung, so dass hier keine Antwort von der Anwendung erfolgen muss.
- Um eine korrekte Bedienung der externen Funkempfänger zu gewährleisten, sollte eine laufende Verarbeitung von Fahrkommandos nicht unterbrochen werden. Die externe Anwendung hat daher die Möglichkeit über Gruppenkommandos mehrerer Empfänger gleichzeitig zu bedienen. Danach sollte auf das entsprechende Ergebnis des Fahrkommandos gewartet werden. Wird ein neuer Befehl abgesetzt, so wird das aktuell durchgeführte Kommando abgebrochen und durch das Neue ersetzt. Bei einem Parallelbetrieb von commeo und iveo Anlagen ist darauf zu achten, dass ein Abbruch von Kommandos, die ein Umschalten des Funsystems zur Folge haben, nicht statt finden und dies mit einem „Busy“ signalisiert wird.
- Alle Variablen und Werte werden immer in einem Zusammenschluss **<array>** gesendet, so dass ein einfacher, gleichbleibender XML Kommandoaufbau entsteht.
- Da bei der Kommunikation ausschließlich XML gesprochen und erwartet wird, kann der standardisierte XML Header bei dem Methodenaufruf durch die externe Anwendung entfallen: ~~**<?xml version=“1.0“? encoding=“UTF-8“>**~~ und minimiert abermals die notwendige Telegramlänge. Bei Antworten, sowie Ereignissen durch das Gateway ist der Header jedoch weiterhin vorhanden, um vorhandene XML Parser in den Anwendungen direkt verwenden zu können.
- Um die SELVE XML Spezifikation auch in kleine Systeme implementieren zu können, ist sie so definiert, dass sämtliche XML Methoden eine Puffergröße von **512 Byte** nicht überschreiten.
- Jedes Element besteht **immer** aus einem Beginn- und einem Ende-Tag (Z.B.: **`<int>`** und **`</int>`**. Die maximale Länge des Inhalts ist entsprechend der einzelnen Methodendefinitionen zu entnehmen und einzuhalten.
- Die **Groß- und Kleinschreibung** ist sowohl bei den Tags, als auch bei den Namen der Methoden zu beachten.
- Aufgrund der Kompatibilität zu den SELVE Produkten und der Sonderbehandlung nach der XML Definition sollten die folgenden Zeichen innerhalb der Zeichenfolgen **`<string>…</string>`** nicht genutzt werden:

  ‘ & ‘ ‘ < ‘ ‘ > ‘ ‘ “ ‘ ‘ ‘ ‘

1.3 XML Tags

Die folgende Tabelle gibt eine Übersicht aller genutzten XML Tags der SELVE XML Spezifikation:

Datentyp	Beschreibung
`<methodCall>`	Beginn eines XML Methodenaufrufs.
`</methodCall>`	Ende eines XML Methodenaufrufs.
`<methodName>`	Beginn des Methodennamens.
`</methodName>`	Ende des Methodennamens.
`<array>`	Beginn des Bereichs, in dem die Variablen und Parameter beschrieben sind.
`</array>`	Ende des Bereichs, in dem die Variablen und Parameter beschrieben sind.
`<int>`	Beginn eines vorzeichenbehafteter Integer, max. Wertebereich -2^{31} bis $(2^{31}-1)$.
`</int>`	Ende eines vorzeichenbehafteter Integer.
`<string>`	Beginn einer Zeichenkette
`</string>`	Ende einer Zeichenkette
`<base64>`	Beginn von Base64-kodierten Binärdaten
`</base64>`	Ende von Base64-kodierten Binärdaten
`<methodResponse>`	Beginn einer Antwort vom Gateway.
`</methodResponse>`	Ende einer Antwort vom Gateway.
`<fault>`	Beginn einer Meldung eines Kommunikationsfehlers.
`</fault>`	Ende einer Meldung eines Kommunikationsfehlers.

Tabelle 1. *Verwendete XML Tags*

1.4 Kommunikationsabläufe

Die folgenden Grafiken zeigt eine korrekte Kommunikation zwischen der Anwendung und dem SELVE USB-RF Gateway.

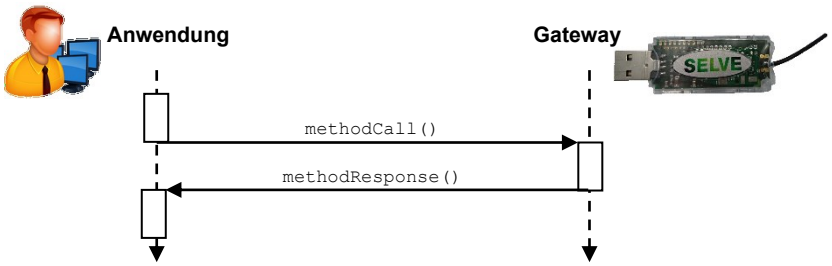

Bild 2. Erfolgreicher Aufruf einer XML Methode durch die Anwendung

Fehlerhafte Aufrufe werden durch ein **<fault>** gemeldet. Die folgende Grafik zeigt schematisch eine Rückmeldung eines Fehlers.

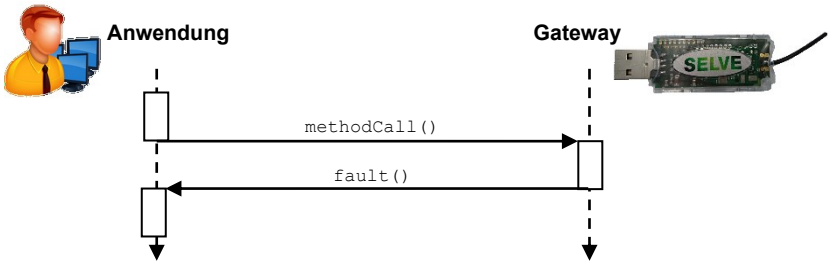

Bild 3. Fehlerhafter Aufruf einer Methode durch die Anwendung

Ergebnisse von Fahrkommandos, etc. werden durch das Gateway automatisch generiert. Diese müssen von der Anwendung **nicht** quittiert werden.

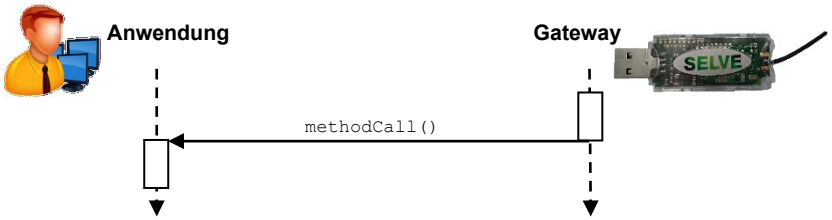

Bild 4. Meldungen und Ergebnisse vom Gateway

1.5 Methodenaufbau

Der grundlegende Aufbau einer Methode, bzw. einer Antwort ist wie folgt definiert.

Startelemente

- **<methodCall>** beschreibt den Start eines Methodenaufruf, welches durch die externe Anwendung erfolgt. Weiterhin erfolgt ein Methodenaufruf in dem Fall, wenn das SELVE USB-RF Gateway automatische Ereignisse, etc. der Anwendung meldet.
- **<methodResponse>** beschreibt den Start einer Antwort durch das SELVE USB-RF Gateway. Diese wird durch den vorherigen Methodenaufruf der Anwendung generiert.

<methodCall>

Ein Methodenaufruf besteht aus zwei wesentlichen Teilen. Als Erstes wird über den Tag **<methodName>** der Methodenname genannt. Werden Parameter (**<int>, <string>, <base64>**) der Methode übergeben, so werden diese in dem darauffolgenden, zusätzlichen Bereich **<array>** in der definierten Reihenfolge hinzugefügt. Wird kein Parameter benötigt, so kann der gesamte Bereich entfallen.

<methodResponse>

Der Aufbau einer Antwort vom USB Stick besteht immer nur aus einem Bereich **<array>**, in dem die Parameter entsprechend der Definition hinzugefügt sind. Eine spezielle Form einer Antwort ist die Fehlermeldung – hier wird das **<array>** in einen **<fault>** Tag gepackt und signalisiert der Anwendung einen entsprechenden Kommunikationsfehler.

Aufbau der XML Methoden

```
<methodCall>
<methodName>Methodenname</methodName>
<array>
    <string>Parameter 1</string>
    <int>Parameter 2</int>
    <base64>Parameter 3</base64>
</array>
</methodCall>
```

```
<?xml version="1.0"?>
<methodResponse>
<array>
    <string>Methodenname</string>
    <string>Parameter 1</string>
    <int>Parameter 2</int>
    <base64>Parameter 3</base64>
</array>
</methodResponse>
```

1.6 Kommunikationsbeispiele

Der folgende Methodenaufruf zeigt eine einfache Anfrage von der Anwendung zum SELVE USB-RF Gateway. In diesem Beispiel wird ein PING gesendet, woraufhin das Gateway mit der entsprechenden Response antwortet.

```
<methodCall>
        <methodName>selve.GW.service.ping</methodName>
</methodCall>
```


```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
        <array>
                <string>selve.GW.service.ping</string>
        </array>
</methodResponse>
```

Das folgende Beispiel zeigt einen fehlerhaften Methodenaufruf, der mit einem Fehler quittiert wird. Als Fehler werden ein String (Fehlerbeschreibung) und eine entsprechende, zugehörige Fehlernummer (Integer) übergeben. Die verschiedenen Fehlermeldungen sind im Anhang A am Ende des Dokuments zusammengefasst.

```
<methodCall>
        <methodName>selve.GW.notSupported</methodName>
        <array>
                <string>Parameter</string>
                <int>100</int>
        </array>
</methodCall>
```

```
<?xml version="1.0"encoding="UTF-8"?>
<methodResponse>
        <fault>
        <array>
                <string>Method not supported!</string>
                <int>2</int>
        </array>
        </fault>
</methodResponse>
```

1.7 Dokumentationsaufbau

Die folgende Darstellung zeigt das allgemein, genutzte Template der folgenden Methodendefinitionen.

Methode:	[Methodenname]
Beschreibung:	[Allgemeine Beschreibung und Informationen zur Methode]
Parameter: <methodCall>	[Beschreibung der einzelnen Parameter, die beim Aufruf **<methodCall>** in der korrekten Reihenfolge in dem **<array>** gesendet werden müssen.]
Parameter: <methodResponse>	[Beschreibung der einzelnen Parameter, die in der Antwort **<methodResponse>** vom SELVE USB-RF Gateway gesendet werden. Die Reihenfolge der Parameter ist entsprechend der Definition.]

Als erstes ist der vollständige Methodenname genannt, der im XML Tag **<methodName>** gesetzt werden muss. Die darauf folgende Beschreibung sorgt für die notwendigen Details der Methode.

Abschließend folgenden die notwendigen Parameter für den Aufruf der Methode **<methodCall>** und die Parameter, die auf der entsprechenden Antwort **<methodResponse>** gesetzt werden.

Hierbei sind die Anzahl und die Reihenfolge der Parameter in dem Datenfeld **<array>** zwingend zu beachten!

2 selve.GW.service – Grundlegende Kommunikation

Die Methodenklasse **selve.GW.service** beinhaltet die Methoden, die für eine grundlegende XML Kommunikation und genereller Einstellungen notwendig sind. Diese Methoden dienen als erstes Grundgerüst für einen ordentlichen Kommunikationsaufbau zwischen der Anwendung und dem Gateway.

In dieser Klasse stehen folgende Methoden der Anwendung zur Verfügung:

Methode	Kurzbeschreibung
`selve.GW.service.ping`	Ping zum SELVE USB-RF Gateway.
`selve.GW.service.getState`	Lesen des aktuellen Status des Gateways.
`selve.GW.service.getVersion`	Lesen der Versionsinformationen des Gateways.
`selve.GW.service.reset`	Neustart des Gateways.
`selve.GW.service.factoryReset`	Das Gateway wird auf den Werkzustand zurückgesetzt.
`selve.GW.service.setLED`	Einstellung der LED Anzeige.
`selve.GW.service.getLED`	Lesen der aktuellen Einstellung der LED Anzeige.

Tabelle 2. *Übersicht der Methodenklasse „selve.GW.service“*

2.1 selve.GW.service.ping

Methode:	`selve.GW.service.ping`
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.ping“ wird das SELVE USB-RF Gateway auf einfache Weise angesprochen. Dieser Befehl ist ohne Parameter entsprechend kurz, so dass mit Hilfe der Methode ein schnelles, automatisiertes Auffinden des Gateways möglich ist. Als Initialisierungsbefehl kann hier z.B. die Anwendung im Falle mehrerer, vorhandener COM Ports das Gateway schnellstmöglich lokalisieren, ohne dass der Anwender einen COM Port vordefinieren muss.
Parameter: <methodCall>	---
Parameter: <methodResponse>	`<string>` Methodenname **`selve.GW.service.ping`**

2.2 selve.GW.service.getState

Methode:	**selve.GW.service.getState**
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.getState“ wird der aktuelle Status des Gateways gelesen. Befindet sich der USB Stick z.B. im Bootloader sind nur eingeschränkt Methoden erreichbar – siehe entsprechendes Kapitel unten. Damit die Anwendung einwandfrei funktioniert sollte eine Anwendung stets warten, bis der Status „Ready“ erreicht ist. Während der „StartUp“ Phase sorgt das Gateway für ein automatisierter Abgleich der internen Informationen eingelernter Aktore, so dass am Ende alle verfügbaren Daten aktualisiert sind. Folgende Statusinformationen werden der Anwendung mitgeteilt: **Bootloader** – Der USB Stick befindet sich im Bootloader. Dieser Bereich ist nach dem Neustart für zwei Sekunden aktiv. Wird kein Firmwareupdate durch die Anwendung durchgeführt und wechselt der Zustand nicht, so signalisiert dies eine mögliche fehlerhafte Firmware. In diesem Fall muss ein neues Firmwareupdate gestartet werden. **Update** – Der USB Stick befindet sich im Bootloader und es wird aktiv ein Firmwareupdate durchgeführt. Ein Update muss durch die Anwendung komplett durchgeführt werden, bevor die eigentliche Firmware wieder lauffähig ist. Ein abgebrochenes, bzw. unterbrochenes Firmwareupdate kann jedoch aus dem Bootloader jederzeit neu gestartet werden. **StartUp** – Die eigentliche Firmware ist gestartet und die StartUp Phase läuft. In dieser Zeit holt sich das Gateway alle aktuellen Zustände der eingelernten Aktore. Je nach Anlagengröße kann die StartUp Phase unterschiedliche Zeiten in Anspruch nehmen. **Ready** – Ist die StartUp Phase beendet, so ist das Gateway betriebsbereit und kann verwendet werden. Alle erhaltenen Akturzustände sind auf Stand und können direkt ausgelesen und verwendet werden. Der Ereignismanager ist nun aktiviert und informiert so die Anwendung automatisch über sich ändernde Zustände.
Parameter: <methodCall>	---
Parameter: <methodResponse>	**<string>** Methodenname **selve.GW.service.getState** **<int>** Status **0 = Bootloader** **1 = Update** **2 = StartUp** **3 = Ready**

2.3 selve.GW.service.getVersion

Methode:	**selve.GW.service.getVersion**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.getVersion“ können die aktuellen Versionsinformationen der aktuell installierten Firmware des Gateways gelesen werden. Die Firmwareversion des USB Sticks setzt sich aus 4 Ziffern zusammen, die wie folgt zu interpretieren sind: Part1.Part2.Part3.Revision (z.B. 16.02.03.01) Weiterhin wird die im Stick verwendete Version der SELVE XML Spezifikation angezeigt: Part1.Part2 (z.B. 2.0) **HINWEIS** Die einzelnen Integerwerte liegen im Gateway als HEX Werte vor, die für eine einheitliche, korrekte Darstellung konvertiert werden sollten.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.service.getVersion**
	<int>	VersionPart1	**Aktuell installierte Firmwareversion (HEX formatiert)**
	<int>	VersionPart2	
	<int>	VersionPart3	
	<int>	SpecPart1	**Version der genutzten SELVE XML Spezifikation (HEX formatiert)**
	<int>	SpecPart2	
	<string>	SerialNo	**Seriennummer des SELVE USB RF-Sticks.**
	<int>	Revision	**Revision der Firmwareversion (HEX formatiert)**

2.4 selve.GW.service.reset

Methode:	**selve.GW.service.reset**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.reset“ kann ein Softwarereset durchgeführt werden. Beim Empfang der Methode wird die entsprechende Antwort der Anwendung gesendet und daraufhin neu gebootet. Diese Methode wird z.B. dazu genutzt um ein Firmwareupdate durchführen zu können. Nach einem Reset sollte die Anwendung stets mit den entsprechend Methoden „selve.GW.service.ping“ und/oder „selve.GW.service.getState“ die gewünschte Bereitschaft abwarten.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.service.reset**
	<int>	Return	**0 = Der Softwarereset kann nicht durchgeführt werden.** **1 = Der Softwarereset wird durchgeführt.**

2.5 selve.GW.service.factoryReset

Methode:	**selve.GW.service.factoryReset**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.factoryReset“ kann das SELVE USB-RF Gateway auf den Werkzustand zurückgesetzt werden. Wird dieser Befehl ausgeführt, so werden alle benutzerspezifischen Einstellungen, eingelernten Sender, Aktore, etc. gelöscht.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.service.factoryReset**
	<int>	Return	**0 = Der Werkzustand konnte nicht durchgeführt werden.** **1 = Der Werksresert wurde durchgeführt.**

2.6 selve.GW.service.setLED

Methode:	**selve.GW.service.setLED**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.setLED“ kann die LED Anzeige des SELVE USB-RF Gateways eingestellt werden. Im Werkszustand ist die LED Anzeige aktiv und zeigt folgende Zustände an: **Dauerhaft Rot** Der USB Stick befindet sich im Bootloader und ist nicht betriebsbereit. **Dauerhaft Gelb** Der USB Stick befindet sich in der StartUp Phase, wo alle Aktordaten eingelesen und aktualisiert werden. **Dauerhaft Grün** Der USB Stick ist betriebsbereit und kann verwendet werden. **Roter Impuls** Ein Telegramm wurde vom USB Stick gesendet. **Gelber Impuls** Ein Telegramm wurde vom USB Stick empfangen. **Blinkt Grün** Ist der USB Stick betriebsbereit so werden über eine blinkende, grüne LED interne Warnungen, die durch Aktore gemeldet werden, signalisiert. Diese bedeutet, dass z.B. ein Aktor nicht mehr erreichbar ist, das ein Aktor ein Hindernis erkannt hat, Sensorverlust, etc. Wird über die Methode die LED Anzeige abgeschaltet, so sind die LEDs im normalen Betrieb aus. Die rote LED für den Bootloader, bzw. die gelbe für die StartUp Phase bleiben im Falle eines Firmwareupdates aktiv.		
Parameter: <methodCall>	**<int>**	LEDModus	**0 = LED Anzeige im Normalbetrieb wird abgeschaltet.** **1 = LED Anzeige im Normalbetrieb wird eingeschaltet.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.service.setLED**
	<int>	Return	**0 = Einstellung konnte nicht vorgenommen werden.** **1 = Einstellung wurde erfolgreich übernommen.**

2.7 selve.GW.service.getLED

Methode:	**selve.GW.service.getLED**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.getLED“ kann die aktuelle Einstellung der LED Anzeige gelesen werden. Die verschiedenen Anzeigen der LEDs sind in der Methode „selve.GW.service.setLED“ detailliert beschrieben.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.service.getLED**
	<int>	LEDModus	**0 = LED Anzeige ist abgeschaltet.** **1 = LED Anzeige ist eingeschaltet.**

3 selve.GW.param – Gateway Einstellungen

Die Methodenklasse **selve.GW.param** beinhaltet die Methoden, die für die benutzerspezifischen Funkeinstellungen und der individuellen Einstellung des Ereignismanagers notwendig sind. Gleichzeitig bietet diese Klasse die Informationen aller internen Funkeinstellungen.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.param.setForward`	Setzen der **commeo** spezifischen Forwarding Einstellungen.
`selve.GW.param.getForward`	Lesen der **commeo** spezifischen Forwarding Einstellungen.
`selve.GW.param.setEvent`	Konfiguration des Ereignismanagers.
`selve.GW.param.getEvent`	Lesen der aktuellen Konfiguration des Event-Managers.
`selve.GW.param.getDuty`	Lesen der aktuellen Funk-Ressourcenausnutzung des Gateways.
`selve.GW.param.getRF`	Lesen der Funkinformationen des Gateways.

Tabelle 3. *Übersicht der Methodenklasse „selve.GW.param“*

3.1 selve.GW.param.setForward

Methode:	`selve.GW.param.setForward`		
Beschreibung:	Mit Hilfe der Methode „selve.GW.param.setForward“ kann das commeo Forwarding im Gateway aktiviert, bzw. deaktiviert werden. Bei aktiviertem Forwarding bedeutet dies, dass das Gateway als zusätzliches Routing Element in einem commeo System agiert. Je nach Anlagensituation kann diese Funktion im Gateway ausgeschaltet werden, da z.B. der Installationsort des Gateways dagegen spricht. Im Werkszustand ist diese Funktion stets eingeschaltet.		
Parameter: <methodCall>	**<int>**	Forwarding	**0 = Das Forwarding wird abgeschaltet.** **1 = Das Forwarding wird eingeschaltet.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.setForward**
	<int>	Return	**0 = Die Einstellung wurde nicht übernommen.** **1 = Die Einstellung wurde übernommen.**

3.2 selve.GW.param.getForward

Methode:	`selve.GW.param.getForward`		
Beschreibung:	Mit Hilfe der Methode „selve.GW.param.getForward“ kann die aktuelle Einstellung des commeo Forwarding vom Gateway gelesen werden.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.getForward**
	<int>	Forwarding	**0 = Das Forwarding ist abgeschaltet.** **1 = Das Forwarding ist eingeschaltet.**

3.3 selve.GW.param.setEvent

Methode:	**selve.GW.param.setEvent**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.param.setEvent“ kann der Ereignismanager des Gateways individuell eingestellt werden. Aktivierte Ereignisse sorgen dafür, das Änderungen das Gateway automatisch der Anwendung mitteilt, so dass die Anwendung kein Polling von Statusinformationen durchführen muss. **EventDevice** Ist die Einstellung aktiviert, so sendet das Gateway automatisch alle Zustandsänderungen der eingelernten Aktore. **EventSensor** Ist die Einstellung aktiviert, so teilt das Gateway der Anwendung veränderte Sensordaten automatisch mit. **EventSender** Ist die Einstellung aktiviert, so meldet das Gateway automatisch eingehende Betätigungen eingelernter Sender der Anwendung. **Logging** Ist die Einstellung aktiviert, so sendet das Gateway mögliche Logs der externen Anwendung. Das Gateway selbst besitzt keinen Speicher für Logs, so dass dies die Anwendung an dieser Stelle übernehmen muss. **EventDuty** Ist die Einstellung aktiviert, so sendet das Gateway automatisch Veränderungen der Funkressourcennutzung.		
Parameter: <methodCall>	**<int>**	EventDevice	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	EventSensor	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	EventSender	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	Logging	**0 = Logs werden nicht gesendet.** **1 = Logs werden gesendet.**
	<int>	EventDuty	**0 = Sich ändernde Funkressourcen-nutzung werden nicht gesendet.** **1 = Sich ändernde Funkressourcen-nutzung werden gesendet.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.setEvent**
	<int>	Return	**0 = Die Einstellungen wurde nicht übernommen.** **1 = Die Einstellungen wurden übernommen.**

3.4 selve.GW.param.getEvent

Methode:	**selve.GW.param.getEvent**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.param.getEvent“ können die Einstellungen des Ereignismanagers des Gateways gelesen werden. Detaillierte Beschreibungen sind bei der Methode „selve.GW.param.setEvent“ aufgeführt.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.getEvent**
	<int>	EventDevice	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	EventSensor	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	EventSender	**0 = Ereignisse werden nicht gesendet.** **1 = Ereignisse werden gesendet.**
	<int>	Logging	**0 = Logs werden nicht gesendet.** **1 = Logs werden gesendet.**
	<int>	EventDuty	**0 = Sich ändernde Funkressourcen-nutzung werden nicht gesendet.** **1 = Sich ändernde Funkressourcen-nutzung werden gesendet.**

3.5 selve.GW.param.getDuty

Methode:	**selve.GW.param.getDuty**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.param.getDuty“ kann die aktuelle Funkressourcennutzung des Gateways gelesen werden. Aufgrund des genutzten ISM-Bands 868 MHz sorgt das Gateway selbständig dafür die geforderten Richtlinien einzuhalten. Die hier gelesene Auslastung wird in Prozent mitgeteilt. Dieser Wert gibt an, wie groß die aktuelle Auslastung bezogen auf des erlaubten Grenzwerts ist. Eine 100 %ige Auslastung bedeutet, dass der Grenzwert, der nach der Richtlinie erlaubten Funkauslastung, erreicht wurde.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.getDuty**
	<int>	Mode	**0 = Senden wird nicht blockiert.** **1 = Senden wird bei 100 % Auslastung blockiert.**
	<int>	Auslastung	**Prozentualle Funkauslastung nach der Funkrichtlinie (0 bis 100 %)**

3.6 selve.GW.param.getRF

Methode:	**selve.GW.service.getRF**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.service.getRF“ können die internen Funkinformationen des Gateways gelesen werden.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.param.setRF**
	<int>	Network	**commeo Netzwerkadresse**
	<int>	ResetCount	**commeo Resetcounter**
	<int>	RFBaseID	**commeo Funkadresse**
	<int>	SensorNetwork	**commeo Netzwerkadresse für die Sensorsimulation**
	<int>	RFSensorID	**commeo Funkadresse für die Sensorsimulation**
	<int>	IveoResetCount	**iveo Resetcounter**
	<int>	RFIveoID	**iveo Funkadresse**

4 selve.GW.device – Aktorverwaltung

Die Methodenklasse **selve.GW.device** beinhaltet alle Methoden, die für das Einlernen, bzw. Auslernen von **commeo** Aktore, bzw. Motore nötig sind.

Weiterhin erlauben die entsprechenden Methoden die Aktorinformationen und Zustände zu lesen.

Es können bis zu 64 Aktore in dem Gateway eingelernt und verwaltet werden. Jeder eingelernte Aktor bekommt eine feste AktorID (zwischen 0 und 63) zugewiesen. Mit dieser AktorID werden die entsprechenden Daten abgerufen und verwaltet.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.device.scanStart`	Startet einen neuen Suchlauf von **commeo** Aktore und Motore.
`selve.GW.device.scanStop`	Abbrechen, bzw. Beenden des aktuellen Suchvorgangs.
`selve.GW.device.scanResult`	Liefert den aktuellen Fortschritt des Suchvorgangs.
`selve.GW.device.save`	Speichert eine AktorID permanent in das Gateway ein.
`selve.GW.device.getIDs`	Lesen einer Maske, die zeigt welche AktorIDs aktiv genutzt werden.
`selve.GW.device.getInfo`	Lesen einzelner Aktorinformationen
`selve.GW.device.getValues`	Lesen der aktuellen Werte des Aktors (Position, Fahrtrichtung, etc.)
`selve.GW.device.setFunction`	Aufruf von Sonderfunktion der Aktore.
`selve.GW.device.setLabel`	Setzen des Namens eines Aktors.
`selve.GW.device.setType`	Ändern des Typs eines Aktors.
`selve.GW.device.delete`	Löscht einen Aktor aus dem Gateway.
`selve.GW.device.writeManual`	Manuelles Schreiben eines kompletten Aktors.

Tabelle 4. *Übersicht der Methodenklasse „selve.GW.device“*

4.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen um Aktore im Gateway hinzuzufügen, bzw. zu löschen.

Weiterhin ist es möglich über die Schnittstelle Endlagen eines Aktors einzustellen.

HINWEIS **Endlageneinstellungen sollten nur von gesicherten Funkstellen und vom Fachpersonal eingestellt werden. Fehlerhafte Bedienung, Funkstörungen und/oder systenbedingte, zeitliche Telegramm-verzögerungen können zur Zerstörung der Anlage führen!**

4.1.1 Einrichten neuer Aktore

Um neue Aktore in dem Gateway einlernen zu können, muss zunächst ein Suchlauf gestartet werden. Die hier neu gefundenen Aktore können im zweiten Schritt in das Gateway eingelernt werden.

Hier sind folgende Schritte durchzuführen:

- Die Aktore müssen zunächst durch einen bereits angelernten Handsender, oder durch eine Netztrennung in den Installationsmodus geführt werden. Dieser Modus ermöglicht das Auffinden der Aktore innerhalb der nächsten 4 Minuten.
- Ein neuer Suchvorgang wird gestartet. Mit jedem Start eines Suchlaufs werden die Ergebnisse einer bereits durchgeführen Suche zuvor gelöscht.
- Neu gefundene Aktore werden in dem Suchergebnis aufgenommen und erhalten den Speicherzustand „Temporär“.
- Der Fortschritt des Suchvorgangs wird der Anwendung automatisch mitgeteilt, so dass die Anwendung das Ergebnis eines Suchvorgangs abwarten und gleichzeitig den Fortschritt anzeigen kann.
- Das Ergebnis des Suchlaufs wird der Anwendung mitgeteilt, woraufhin die Anwendung die entsprechenden Daten auslesen kann.
- Durch den Aufruf der Methode „speichern“ wird der ausgewählte Aktor fest in das Gateway eingespeichert. Während des Speicherns wird eine feste Verbindung zwischen dem Gateway und dem Aktor erzeugt. Der Aktor erhält nach einem erfolgreichen Verbindungsaufbau den Speicherzustand „In Benutzung“.
- Mit Hilfe der zur Verfügung gestellten Methoden können anschließend Name und/oder Typ des Aktors nachgestellt werden.
- Ein Suchlauf kann jederzeit gestoppt werden.
- Wird das Einrichten neuer Aktore von der Anwendung beendet, so können durch das Senden der Stop-Methode alle noch temporär gekennzeichneten Aktore freigegeben werden. Dadurch wird die Liste neuer, ungespeicherter Aktore ordentlich gelöscht.

Das folgende Ablaufdiagramm zeigt das vollständige Einrichten neuer Aktore.

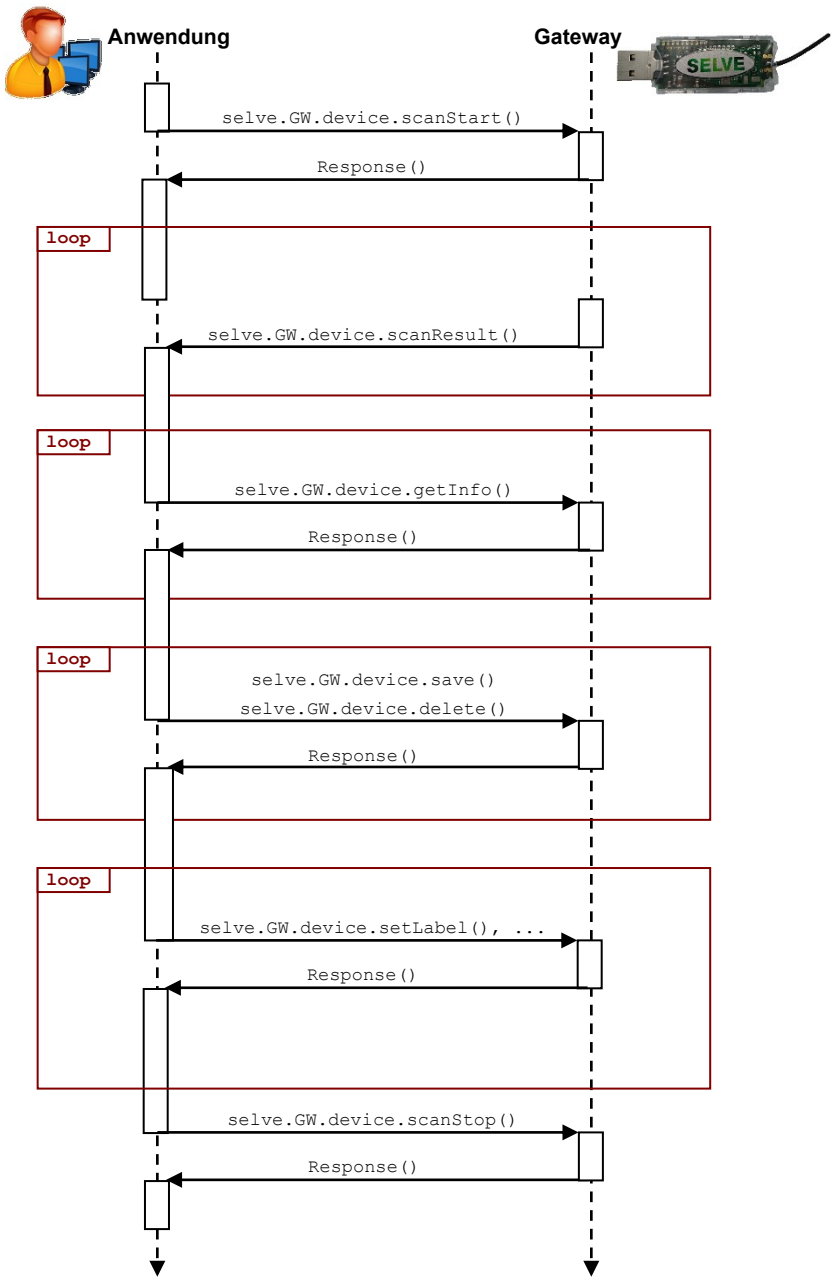

Bild 5. Grafische Darstellung des Ablaufs zum Einrichten neuer Aktore.

4.1.2 Löschen von Aktore

Das Löschen von Aktore kann zu jedem Zeitpunkt durchgeführt werden. Durch den Aufruf der entsprechenden Methode zum Löschen wird der Aktor aus dem Gateway gelöscht.

HINWEIS **Um die Verbindung zwischen dem Aktor und dem Gateway ordentlich zu trennen empfielt es sich das Löschen eines Aktors in Funkreichweite zum Aktor durchzuführen. Dies ermöglicht dem Gateway eine saubere Trennung.**

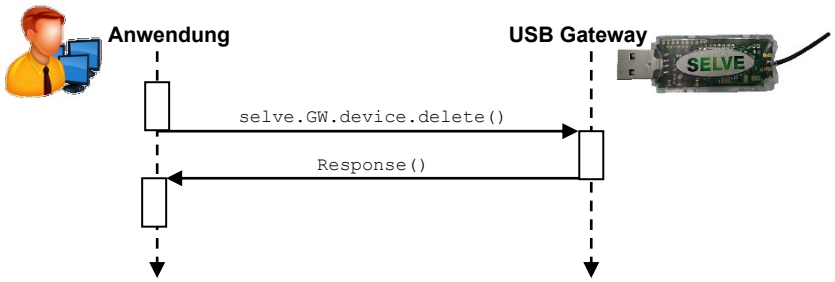

Bild 6. Grafische Darstellung des Ablaufs zum Löschen eingerichteter Aktore.

4.1.3 Ablauf der Endlageneinstellung

Der folgende Ablauf sorgt für ein korrektes Einstellen der Endlagen der Aktore. Mit Hilfe der Methode **selve.GW.device.setFunction** können die Endlagen von **commeo** Aktore sowohl im automatischen Einstellmodus, als auch im manuellen Einstellmodus eingestellt werden.

HINWEIS **Endlageneinstellungen sollten nur von gesicherten Funkstellen und vom Fachpersonal eingestellt werden. Fehlerhafte Bedienung, Funkstörungen und/oder systenbedingte, zeitliche Telegramm-verzögerungen können zur Zerstörung der Anlage führen!**

HINWEIS **Während einer Endlageneinstellung befindet sich der Motor im Totmannbetrieb, so dass durch das Senden des Fahrkommandos „Key-Release“ der Aktor gestoppt wird. Wird dies aus verschiedensten Gründen nicht, oder zu spät gesendet sorgt der Totmannbetrieb dafür, dass die zuvor gestartete Fahrt unter Umständen die Anlage zerstören kann!**

HINWEIS **Die folgenden Einlernverfahren beschreiben das Einstellen der Endlagen an üblichen commeo Motore. Je nach Aktor kann es leichte Abweichungen zum manuellen, bzw. automatischen Einlernenprozess geben, die der entsprechenden Bedienungsanleitung des Aktors zu entnehmen sind.**

Manuelle Endlageneinstellung

- Um den Aktor visuell auswählen zu können, kann optional mit Hilfe von **selve.GW.device.setFunction(Select)** der Aktor selektiert werden. Der Aktor meldet sich i.d.R. durch ein kurzes Zucken.
- Um das manuelle Einlernen der Endlagen zu starten, wird zunächst der Befehl **selve.GW.device.setFunction(ManProg)** gesendet. Durch das Senden werden bereits eingerichtete Endlagen gelöscht.
- Mit Hilfe der speziellen Fahrkommandios: **setFunction(DriveUp)**, **setFunction(DriveDown)** und **setFunction(KeyRelease)** kann der Aktor nun im Totmannmodus betrieben werden. Hier ist darauf zu achten, dass das „KeyRelease“ nach jeder Ab-, bzw. Auffahrt gesendet wird, damit der Aktor entsprechend stoppt und korrekt arbeitet.
- Mit den Fahrkommandos wird nun zunächst die untere Endlage angefahren. Mit dem Kommando **selve.GW.device.setFunction(StorePosition)** wird die angefahrende Position als untere Endlage gespeichert.
- Im nächsten Schritt wird die obere Endlage angefahren. Mit dem erneuten Senden des Kommandos **selve.GW.device.setFunction(StorePosition)** wird nun die angefahrende Position als obere Endlage gespeichert.
- Das manuelle Einstellen der Endlagen ist beendet und der Motor fährt nun nicht länger im Totmannbetrieb. Die üblichen Fahrkommandos können nun verwendet werden.

Automatische Endlageneinstellung

- Um den Aktor visuell auswählen zu können, kann optional mit Hilfe von **selve.GW.device.setFunction(Select)** der Aktor selektiert werden. Der Aktor meldet sich i.d.R. durch ein kurzes Zucken.
- Um das automatische Einlernen der Endlagen zu starten, wird zunächst der Befehl **selve.GW.device.setFunction(AutoProg)** gesendet. Durch das Senden werden bereits eingerichtete Endlagen gelöscht.
- Mit Hilfe der speziellen Fahrkommandios: **setFunction(DriveUp)**, **setFunction(DriveDown)** und **setFunction(KeyRelease)** kann der Aktor nun im Totmannmodus betrieben werden. Hier ist darauf zu achten, dass das „KeyRelease“ nach jeder Ab-, bzw. Auffahrt gesendet wird, damit der Aktor entsprechend stoppt und korrekt arbeitet.
- Zunächst wird der untere Punkt des Aktors angefahren, damit dieser Prozess, unabhängig vom commeo Gerät funktioniert.
- Für das automatische Einlernverfahren muss nun der Aktor lediglich in die obere Endlage gefahren werden. Je nach Aktortyp reversiert er selbstständig und sucht sich anschließend automatisch den unteren Endpunkt.
- Ist der untere Endpunkt gefunden, so ist das Einstellen der Endlagen beendet und der Motor fährt nun nicht länger im Totmannbetrieb. Die üblichen Fahrkommandos können nun wieder verwendet werden.

4.2 selve.GW.device.scanStart

Methode:	**selve.GW.device.scanStart**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.scanStart“ wird ein Suchlauf nach neuen Aktoren gestartet. Eine vorherige Liste neuer Aktore wird mit diesem Aufruf gelöscht, so dass alle zu diesem Zeitpunkt nicht eingelernten Aktore aus dem Gateway verschwinden. Ist bereits ein Suchlauf gestartet, wird dieser automatisch abgebrochen und ein neuer Suchlauf gestartet.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.scanStart**
	<int>	Return	**0 = Ein neuer Suchlauf konnte nicht gestartet werden.** **1 = Ein neuer Suchlauf ist gestartet worden.**

4.3 selve.GW.device.scanStop

Methode:	**selve.GW.device.scanStop**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.scanStop“ kann ein laufender Suchlauf gestoppt werden. Bei einem Aufruf nach einem erfolgreichen Suchlauf sorgt diese Methode dafür, dass alle temporär eingerichteten Aktore gelöscht werden und somit das Gateway bereinigt wird.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.scanStop**
	<int>	Return	**0 = Ein interner Fehler ist aufgetreten.** **1 = Der Suchlauf ist gestoppt und die Liste temporärer Aktore ist bereinigt.**

4.4 selve.GW.device.scanResult

Methode:	**selve.GW.device.scanResult**
Beschreibung:	**HINWEIS Diese Methode wird als Ergebnis automatisch vom Gateway erzeugt und meldet zyklisch den Fortschritt eines Suchlaufs.** **Alternativ kann die gleiche Methode auch dazu genutzt werden, um von der Anwendung manuell einen aktuellen Fortschritt anzufragen. In diesem Fall sind bei der Anfrage an das Gateway keine Parameter erforderlich und das Gateway antwortet entsprechend der üblichen Form mit dem Methodennamen und den unten beschrieben Parametern.** Mit Hilfe der Methode „selve.GW.device.scanResult“ wird der Anwendung der aktuelle Status eines Aktor-Suchlaufs mitgeteilt. Ein Suchlauf besitzt dabei verschiedene Teilschritt die wie folgt definiert sind: **Idle** Steht der Zustand auf diesen Wert, so ist kein Suchlauf gestartet. **Run** In diesem Zustand ist der Suchlauf gestartet und es werden neue Aktore gesucht. Die Anzahl neu gefundener und als temporär eingetragene Aktore werden durch die automatische Fortschrittsmeldung der Anwendung mitgeteilt. **Verify** In diesem Zustand ist der Suchlauf abgeschlossen und die Daten der gefundenen Aktore werden gelesen und deren Funkverbindung geprüft. Aktore, die an dieser Stelle funktechnisch nicht erreichbar sind, werden aus der Liste neugefundener Aktore herrausgenommen. **End_Success** Dieser Zustand signalisiert der Anwendung einen erfolgreich durchgeführten Suchlauf. Die mitgelieferte Maske zeigt die AktorIDs der neu gefundenen Aktore für einen optimierten Download der Daten an. **End_Failed** Dieser Zustand signalisiert einen internen Fehler während eines Suchlaufs. Die mitgelieferte Maske der AktorIDs ist genau 8 Bytes lang. Jedes Bit repräsentiert eine AktorID. Ist das Bit auf 1 gesetzt, so handelt es sich bei dieser AktorID um einen neu gefundenen Aktor. Eine 0 zeigt, dass diese AktorID entweder bereits genutzt wird, oder es sich um eine freie, nicht genutzte AktorID handelt. Byte 0.0 AktorID 0 ... Byte 0.7 AktorID 7 Byte 1.0 AktorID 8 ... Byte 7.7 AktorID 63
Parameter: <methodCall>	**<int>** Status **0 = Idle** **1 = Run** **2 = Verify** **3 = End_Success** **4 = End_Failed** **<int>** NoOfNewDevices **Aktuelle Anzahl neu gefundener Aktore.** **<base64>** Maske **Maske der neu gefundenen AktorIDs**
Parameter: <methodResponse>	---

4.5 selve.GW.device.save

Methode:	**selve.GW.device.save**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.save“ wird ein Aktor mit der ausgewählten AktorID permanent im Gateway gespeichert. An dieser Stelle sorgt das Gateway für einen automatisierten Einlernprozess und den Abgleich aller relevanten Daten, so dass der Aufruf eine kurze Zeit andauert.		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der permanent eingelernt werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.save**
	<int>	Return	**0 = Der Einlernprozess ist fehlgeschlagen.** **1 = Der Einlernprozess war erfolgreich.**

4.6 selve.GW.device.getIDs

Methode:	**selve.GW.device.getIDs**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.getIDs“ kann eine Maske gelesen werden, die zeigt welche AktorIDs im Gateway aktuell in Benutzung sind. Diese Methode ermöglicht einen optimierten Zugriff. Wird z.B. während der Initialisierungphase einer Anwendung alle Daten erfasst, so kann durch diese Maske das Lesen der notwendigen Aktordaten beschleunigt werden. Die Maske der AktorIDs ist genau 8 Bytes lang. Jedes Bit repräsentiert eine AktorID. Ist das Bit auf 1 gesetzt, so handelt es sich hierbei um eine genutzte AktorID. Eine 0 zeigt, dass diese AktorID nicht genutzt wird und bei z.B. einem kompletten Download nicht relevant ist. Byte 0.0 AktorID 0 ... Byte 0.7 AktorID 7 Byte 1.0 AktorID 8 ... Byte 7.7 AktorID 63		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.getIDs**
	<base64>	Maske	**Maske der genutzten AktorIDs**

4.7 selve.GW.device.getInfo

Methode:	**selve.GW.device.getInfo**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.getInfo“ werden die aktuell vorliegenden Informationen der ausgewählten AktorID gelesen. Zu dem aktuellen Namen des Aktors und dem Aktortyp wird auch die Funkadresse und der aktuelle Speicherzustand mitgeteilt. Diese Daten sind für ein mögliches Backup der Anwendung relevant.		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) von dem die Daten gelesen werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.getInfo**
	<int>	AktorID	**AktorID zu dem die folgenden Daten gehören (0 bis 63)**
	<int>	Funkadresse	**Funkadresse des Aktors**
	<string>	Aktorname	**Aktueller Name des Aktors (UTF-8 formatiert).**
	<int>	Konfiguration	**0 = Unbekannte Konfiguration** **1 = Rollladen** **2 = Jalousie** **3 = Markise** **4 = Schaltaktor** **5 = Dimmer** **6 = Nachtlicht Aktor** **7 = Dämmerlicht Aktor** **8 = Heizung** **9 = Kühlgerät** **10 = Schaltaktor (Tagbetrieb)** **11 = Gateway**
	<int>	Status	**0 = AktorID wird nicht genutzt** **1 = AktorID wird genutzt** **2 = AktorID wird temporär genutzt** **3 = AktorID wurde früher genutzt. Durch ein Löschvorgang wird sie derzeitig nicht mehr genutzt.**

4.8 selve.GW.device.getValues

Methode: **selve.GW.device.getValues**

Beschreibung:

Mit Hilfe der Methode „selve.GW.device.getValues“ werden die aktuell vorliegenden Zustände des ausgewählten Aktors gelesen.

Status Dieser Wert beschreibt den aktuellen Status des Aktors und zeigt z.B. ob ein Motor gerade auffährt, abfährt oder steht.

Position Dieser Wert beschreibt die aktuelle Position des entsprechenden Aktors. Dieser Wert kann wie folgt interpretiert werden:

```
    0     = Obere Endlage eines Motors, bzw.
            Aktor ist ausgeschaltet
... 65535 = Untere Endlage eines Motors, bzw.
            Aktor ist eingeschaltet.
```

Flags Der Wert beschreibt eine Bit-Maske, die wie folgt zu interpretieren ist.

```
Bit 0 - 1 = Aktor kann nicht mehr erreicht werden.
Bit 1 - 1 = Aktor meldet „Überlast“.
Bit 2 - 1 = Aktor meldet „Hindernis“.
Bit 3 - 1 = Das „Alarmsignal“ des Aktors ist gesetzt.
Bit 4 - 1 = Aktor meldet „Sensorverlust“.
Bit 5 - 0 = Aktor steht auf „Manuell“, automatische
            Fahrten werden nicht ausgeführt.
        1 = Automatik des Aktors ist aktiv.
Bit 6 - 1 = Das Gateway ist im Aktor nicht länger
            eingelernt.
Bit 7 – 1 = Aktor befindet sich im „Windalarm“.
Bit 8 – 1 = Aktor befindet sich im „Regenalarm“.
Bit 9 – 1 = Aktor befindet sich im „Frostalarm“.
```

DayMode Dieser Wert beschreibt den internen Tageszustand, der z.B. durch eine Zeitschaltuhr (commeo MultiSend, etc.) gesetzt wird. Dieser Zustand sorgt für entsprechendes Sensorverhalten im Aktor.

Parameter: <methodCall>

<int>	AktorID	**Nummer des Aktors (0 bis 63) von dem die Daten gelesen werden sollen.**

Parameter: <methodResponse>

<string>	Methodenname	**selve.GW.device.getValues**
<int>	AktorID	**AktorID zu dem die folgenden Daten gehören (0 bis 63)**
<int>	Status	**0 = Unbekannter Zustand** **1 = Aktor ist gestoppt, bzw. Aktor ist ausgeschaltet** **2 = Aktor fährt hoch, bzw. Aktor ist eingeschaltet** **3 = Aktor fährt runter, bzw. Aktor ist eingeschaltet**
<int>	Value	**Aktueller Wert des Aktors (0 bis 65535)**
<int>	TargetValue	**Zielwert des Aktors (0 bis 65535) falls der Aktor in Bewegung ist.**
<int>	Flags	**Maske der Zustandsbits (siehe oben)**
<int>	DayMode	**0 = Unbekannter Tageszustand** **1 = Aktor befindet sich im Nachtmodus** **2 = Aktor befindet sich in der Morgendämmerung** **3 = Aktor befinde sich im Tagmodus** **4 = Aktor befindet sich in der Abenddämmerung**
<string>	Aktorname	**Aktueller Name des Aktors (UTF-8 formatiert).**

4.9 selve.GW.device.setFunction

Methode:	**selve.GW.device.setFunction**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.setFunction“ können Spezialbefehle an den entsprechenden Aktor gesendet werden. Folgende Funktionen stehen hier zur Verfügung:		
	Select	Mit dieser Funktion wird dem Aktor mitgeteilt eine Signalisierungsfahrt durchzuführen. Bei z.B. Motore ist dies ein kurzes Zucken.	
	Install	Mit dieser Funktion wird dem Aktor mitgeteilt den Installationsmodus neu zu starten. Damit kann der Aktor von anderen commeo Sender wieder gefunden werden, ohne ihn vom Netz trennen zu müssen.	
	Sensor	Mit dieser Funktion kann der Aktor in den Sensorlernmodus versetzt werden, um einen Sensor im Aktor einlernen zu können.	
	ManProg	Die manuelle Endlageneinstellung im Aktor wird gestartet.	
	AutoProg	Die automatische Endlageneinstellung im Aktor wird gestartet.	
	StorePosition	In der manuellen Endlageneinstellung wird diese Funktion zum Speichern der Endlagen genutzt.	
	DriveUp	Während der Endlageneinstellung kann mit dieser Funktion der Aktor im Totmannbetrieb aufgefahren werden.	
	DriveDown	Während der Endlageneinstellung kann mit dieser Funktion der Aktor im Totmannbetrieb abgefahren werden.	
	KeyRelease	Während der Endlageneinstellung befinden sich die Aktor im Totmannbetrieb, so dass mit dieser Funktion die Fahrt unterbrochen werden muss.	
	DriveStop	Während der Endlageneinstellung kann mit Hilfe eines direkten Stops der Aktor ebenfalls gestoppt werden.	
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der die Funktion ausführen soll.**
	<int>	Funktion	**0 = Select** **1 = Install** **2 = Sensor** **3 = ManProg** **4 = AutoProg** **5 = StorePosition** **6 = DriveUp** **7 = DriveDown** **8 = KeyRelease** **9 = DriveStop**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.setFunction**
	<int>	Return	**0 = Funktion konnte nicht aufgerufen werden.** **1 = Funktion wurde aufgerufen.**

4.10 selve.GW.device.setLabel

Methode:	**selve.GW.device.setLabel**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.setLabel“ kann der Name des Aktors geändert werden. Dieser Name wird nicht nur im Gateway hinterlegt, sondern per Funk direkt im Aktor gespeichert.		
	HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der den Namen erhalten soll.**
	<string>	Name	**Neuer Name des Aktors (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.setLabel**
	<int>	Return	**0 = Der Name konnte nicht übernommen werden.**
			1 = Der Name konnte erfolgreich im Aktor geändert werden.

4.11 selve.GW.device.setType

Methode:	**selve.GW.device.setType**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.setType“ kann die grundlegende Konfiguration im Aktor verändert werden.		
	HINWEIS Nicht alle Aktore besitzen alle Konfigurationsmöglichkeiten! So gibt es z.B. einen Fehler, wenn ein Rollladenmotor als Schaltaktor konfiguriert werden soll.		
	Die verschiedenen Konfigurationsmöglichkeiten entnehmen Sie den Bedienungsanleitungen der entsprechenden Geräte!		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der die Konfiguration erhalten soll.**
	<int>	Konfiguration	**1 = Rollladen**
			2 = Jalousie
			3 = Markise
			4 = Schaltaktor
			5 = Dimmer
			6 = Nachtlicht Aktor
			7 = Dämmerlicht Aktor
			8 = Heizung
			9 = Kühlgerät
			10 = Schaltaktor (Tagbetrieb)
			11 = Gateway
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.setType**
	<int>	Return	**0 = Die Konfiguration konnte im Aktor nicht übernommen werden.**
			1 = Die Konfiguration konnte erfolgreich im Aktor geändert werden.

4.12 selve.GW.device.delete

Methode:	**selve.GW.device.delete**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.delete“ kann ein Aktor aus dem Gateway gelöscht werden. Das Gateway führt parallel einen automatisierten Auslernprozess durch. Ist dieser erfolgriech, so wird die AktorID für zukünftige Einlernprozesse wieder freigegeben. Findet das Löschen ohne direkte Funkkommunikation zum Aktor statt, so wird diese ebenfalls aus dem Gateway ausgetragen und als „Nicht mehr genutzt“ (Status = 3) gekennzeichnet.		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.delete**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Der Aktor konnte erfolgreich aus dem Gateway entfernt werden.**

4.13 selve.GW.device.writeManual

Methode:	**selve.GW.device.writeManual**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.device.writeManual“ kann ein Aktor manuell im Gateway hinzugefügt werden. Diese Methode kann im Falle eines Backups genutzt werden. Folgende Randbedinungen sind dabei zu erfüllen, damit das manuelle Einrichten des Aktors durchgeführt wird: 1. Die gewählte AktorID darf nicht bereits vom Gateway genutzt werden. Nur bei nicht genutzter AktorID erlaubt das Gateway ein manuelles Einrichten. 2. Die Funkadresse des Aktors darf im Gateway noch nicht hinter einer anderen AktorID vorliegen. 3. Der Aktor muss in Funkreichweite sein, da während des Einrichtens der Aktor eingelernt und die entsprechenden Daten abgeglichen werden. **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**		
Parameter: <methodCall>	**<int>**	AktorID	**AktorID, die für das manuelle Einrichten genutzt wird (0 bis 63).**
	<int>	Funkadresse	**Funkadresse des Aktors (0x000000 bis 0x7FFFFF)**
	<string>	Name	**Neuer Name des Aktors (Maximal 23 Byte)**
	<int>	Konfiguration	**1 = Rollladen** **2 = Jalousie** **3 = Markise** **4 = Schaltaktor** **5 = Dimmer** **6 = Nachtlicht Aktor** **7 = Dämmerlicht Aktor** **8 = Heizung** **9 = Kühlgerät** **10 = Schaltaktor (Tagbetrieb)** **11 = Gateway**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.device.writeManual**
	<int>	Return	**0 = Das manuelle Einrichten des Aktors ist fehlgeschlagen.** **1 = Der Aktor konnte erfolgreich im Gateway eingerichtet werden.**

5 selve.GW.sensor – Sensorverwaltung

Die Methodenklasse **selve.GW.sensor** beinhaltet alle Methoden, die für das Einlernen, und der Verwaltung von **commeo** Sensoren notwendig sind.

Die entsprechenden Methoden erlauben das Lesen der Sensorinformationen und deren aktuellen Zustände.

Es können bis zu 8 **commeo** Sensoren in das Gateway eingelernt und verwaltet werden. Jeder Sensor bekommt eine feste SensorID (zwischen 0 und 7) zugewiesen. Mit dieser SensorID können die entsprechenden Daten abgerufen und verwaltet werden.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.sensor.teachStart`	Startet einen neuen Einlernprozess eines **commeo** Sensors.
`selve.GW.sensor.teachStop`	Stoppt einen laufenden Sensor-Einlernprozess.
`selve.GW.sensor.teachResult`	Liefert den aktuellen Fortschritt des Sensor-Einlernprozesses.
`selve.GW.sensor.getIDs`	Lesen einer Bitmaske, die beschreibt welche SensorIDs aktuell genutzt werden.
`selve.GW.sensor.getInfo`	Auslesen der Sensorinformationen
`selve.GW.sensor.getValues`	Lesen der aktuellen Werte des Sensors
`selve.GW.sensor.setLabel`	Setzen eines Labels für den Sensor.
`selve.GW.sensor.delete`	Löscht einen Sensor aus der Liste
`selve.GW.sensor.writeManual`	Manuelles schreiben eines Sensors in die Liste

Tabelle 5. *Übersicht der Methodenklasse „selve.GW.sensor“*

5.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen um Sensoren im Gateway hinzuzufügen, bzw. zu löschen.

5.1.1 Einlernen von Sensoren

Das Einlernen eines Sensors erfolgt in den folgenden Schritten

- Um einen Sensor einlernen zu können wird über die Methode **selve.GW.sensor.teachStart** der Einlernprozess im Gateway gestartet. Der Einlernprozess ist nach Aufruf der Methode für 4 Minuten aktiv und schaltet sich, wenn kein Sensor eingelernt wird, automatisch nach dieser Zeit ab. Die Methode **selve.GW.sensor.teachResult** liefert an dieser Stelle zyklisch einen Status, der die noch zur Verfügung stehende Zeit im Einlernprozess mitteilt.
- Für das Einlernen wird am entsprechenden **commeo** Sensor die PROG-Taste kurz betätigt. Durch das Drücken wird das Einlernen im Gateway automatisch durchgeführt.
- Über die Methode **selve.GW.sensor.teachResult** wird das erfolgreiche Einlernen des Sensors automatisch der Anwendung mitgeteilt und der Einlernprozess an dieser Stelle beendet. Für das Einlernen mehrerer Sensoren muss der Einlernprozess neu gestartet werden.

5.1.2 Löschen von Sensoren

Das Löschen von Sensoren kann zu jedem Zeitpunkt durchgeführt werden. Durch den Aufruf der entsprechenden Methode wird die Verbindung des Sensors im Gateway gelöscht und steht der Anwendung nicht mehr zur Verfügung.

Wird im Nachhinein ein neuer Sensor eingelernt, so kann dieser den freien SensorID Platz einnehmen!

5.1.3 Sensorvariablen

In diesem Abschnitt werden die möglichen Sensordaten eines Sensors detailliert beschrieben. Die darauf folgenden Methoden in der Klasse selve.GW.sensor, selve.GW.senSim und selve.GW.event beinhalten identische Definitionen der Sensordaten.

Sensorvariable	Beschreibung

`WindDigital`	`0 = Der Sensor besitzt keinen Windsensor, oder es gibt noch keine gültigen Daten.`
	`1 = Der Sensor besitzt einen Windsensor. Die eingestellte Windschwelle ist noch nicht erreicht, so dass kein Windalarm vorliegt.`
	`2 = Der Sensor besitzt einen Windsensor. Die eingestellte Windschwelle ist erreicht, so dass ein Windalarm vorliegt.`

RainDigital	0 = Der Sensor besitzt keinen Regensensor, oder es gibt noch keine gültigen Daten. 1 = Der Sensor besitzt einen Regensensor. Derzeit liegt am Sensor kein Regen an. 2 = Der Sensor besitzt einen Regensensor. Derzeit erkennt der Sensor Regen, so dass ein Regenalarm vorliegt.
TempDigital	0 = Der Sensor besitzt keinen Temperatursensor, oder es gibt noch keine gültigen Daten. 1 = Es herrscht eine normale Umgebungstemperatur von > 4.0°C und ≤ 20.0 °C. 2 = Der Sensor meldet „Frost“ (Temperaturen ≤ 4.0 °C). 3 = Der Sensor meldet „Warm“ (Temperaturen > 20.0 °C).
LightDigital	0 = Der Sensor besitzt keinen Lichtsensor, oder es gibt noch keine gültigen Daten. 1 = Der Lichtsensor meldet den Zustand „Dunkel“. 2 = Der Lichtsensor meldet den Zustand „Dämmerig“. 3 = Der Lichtsensor meldet normale Helligkeit (Standardwert). 4 = Der Lichtsensor meldet den Zustand „Sonnig“.
TempAnalog	Ein Sensor mit Temperaturmessung kann den analogen Temperaturwert melden. Dieser ist für den Bereich von -40.0 °C bis +85.0 °C in 0.5 °C Schritten normiert. -1 = Der analoge Temperaturwert ist nicht verfügbar. 0 = -40.0 °C ... 80 = 0.0 °C ... 250 = +85.0 °C
WindAnalog	Ein Sensor mit Windgeschwindigkeitsmessung kann den analogen Windwert melden. Dieser ist für den Bereich von 0.0 m/s bis 50.0 m/s in 0.5 m/s Schritten normiert. -1 = Der analoge Windwert ist nicht verfügbar. 0 = 0.0 m/s ... 100 = 50.0 m/s
Sun1Analog **Sun2Analog** **Sun3Analog**	Ein Sensor mit Sonnenmessung kann den analogen Sonnenwert melden. Dieser ist für den Bereich von 0 kLux bis 125 kLux in 0.5 kLux Schritten normiert. -1 = Der analoge Sonnenwert ist nicht verfügbar. 0 = 0 kLux ... 250 = 125 kLux
DayLightAnalog	Ein Sensor mit Tageslichtmessung kann den analogen Tageslichtwert melden. Dieser ist für den Bereich von 0 Lux bis 1000 Lux in 4 Lux Schritten normiert. -1 = Der analoge Tageslichtwert ist nicht verfügbar. 0 = 0 Lux ... 250 = 1000 Lux

Tabelle 6. *Übersicht der definierten Sensorvariablen.*

5.2 selve.GW.sensor.teachStart

Methode:	**selve.GW.sensor.teachStart**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.teachStart“ wird der Sensoreinlernmodus für 4 Minuten im Gateway gestartet. Sind bereits 8 Sensoren eingelernt, so wird der Start eines neuen Einlernprozesses unterbunden und mit einem Fehler quittiert. Ist bereits ein Einlernprozess gestartet, wird dieser automatisch gestoppt und einer neuer gestartet.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.teachStart**
	<int>	Return	**0 = Der Einlernprozess konnte nicht gestartet werden.** **1 = Der Einlernprozess ist gestartet.**

5.3 selve.GW.sensor.teachStop

Methode:	**selve.GW.sensor.teachStop**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.teachStop“ wird der Sensoreinlernprozess gestoppt.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.teachStop**
	<int>	Return	**0 = Der Einlernprozess konnte nicht gestoppt werden.** **1 = Der Einlernprozess ist gestoppt worden.**

5.4 selve.GW.sensor.teachResult

Methode:	**selve.GW.sensor.teachResult**
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt und meldet zyklisch den Fortschritt eines Sensoreinlernprozesses.** **Alternativ kann die gleiche Methode auch dazu genutzt werden, um von der Anwendung manuell einen aktuellen Status des Einlernprozesses anzufragen. In diesem Fall sind bei der Anfrage an das Gateway keine Parameter erforderlich und das Gateway antwortet entsprechend der üblichen Form mit dem Methodennamen und den unten beschrieben Parametern.** Mit Hilfe der Methode „selve.GW.sensor.teachResult“ wird der Anwendung der aktuelle Status eines Sensoreinlernprozesses mitgeteilt. Folgende Zustände werden über den Status mitgeteilt: **Idle** Steht der Zustand auf diesen Wert, so ist kein Einlern-prozess gestartet. **Run** In diesem Zustand ist der Einlernprozess gestartet und ein Sensor kann eingelernt werden. Ist die interne Zeit abgelaufen, so wird der Prozess automatisch beendet und der Prozess wechselt zurück in den „Idle“ Zustand. **End_Success** Dieser Zustand signalisiert der Anwendung dass ein Sensor erfolgreich eingelernt wurde. Die mitgelieferte SensorID zeigt an dieser Stelle den neu eingelernten Sensor.
Parameter: <methodCall>	**<int>** Status **0 = Idle** **1 = Run** **2 = End_Success** **<int>** TimeLeft **Angabe in Sekunden, wie lange der Einlernprozess noch aktiv ist.** **<int>** SensorID **-1 = Kein neuer Sensor verfügbar.** **0 bis 7 = SensorID des neuen Sensors.**
Parameter: <methodResponse>	---

5.5 selve.GW.sensor.getIDs

Methode:	**selve.GW.sensor.getIDs**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.getIDs“ kann eine Maske gelesen werden, die zeigt welche SensorIDs im Gateway aktuell in Benutzung sind. Diese Methode ermöglicht einen optimierten Zugriff. Werden z.B. während der Initialisierungphase einer Anwendung alle Daten erfasst, so kann durch diese Maske das Lesen der notwendigen Sensordaten beschleunigt werden. Die Maske der SensorIDs ist genau 1 Byte lang. Jedes Bit repräsentiert eine SensorID. Ist das Bit auf 1 gesetzt, so handelt es sich bei dieser SensorID um einen genutzten Sensor. Eine 0 zeigt, dass diese SensorID nicht genutzt wird und bei z.B. einem kompletten Download nicht relevant ist. Byte 0.0 SensorID 0 ... Byte 0.7 SensorID 7		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.getIDs**
	<base64>	Maske	**Maske der genutzten SensorIDs**

5.6 selve.GW.sensor.getInfo

Methode:	**selve.GW.sensor.getInfo**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.getInfo“ werden die aktuell vorliegenden Informationen der ausgewählten SensorID gelesen. Zu dem aktuellen Namen wird an dieser Stelle auch die Funkadresse des Sensors und der aktuelle Speicherzustand mitgeteilt. Diese Daten sind für ein mögliches Backup der Anwendung relevant. **HINWEIS SensorIDs mit der Funkadresse 0x000000 sind als unbenutzt zu kennzeichnen.**		
Parameter: <methodCall>	**<int>**	SensorID	**Nummer des Sensors (0 bis 7) von dem die Daten gelesen werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.getInfo**
	<int>	SensorID	**SensorID zu dem die folgenden Daten gehören (0 bis 7)**
	<int>	Funkadresse	**Funkadresse des Sensors (0xD80000 bis 0xEFFFFF)**
	<string>	Sensorname	**Aktueller Name des Sensors (UTF-8 formatiert).**

5.7 selve.GW.sensor.getValues

Methode:	**selve.GW.sensor.getValues**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.getValues“ werden die aktuell vorliegenden Sensordaten des ausgewählten Sensors gelesen.		
Parameter: <methodCall>	**<int>**	SensorID	**Nummer des Sensors (0 bis 7) von dem die Daten gelesen werden sollen.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.getValues**
	<int>	SensorID	**SensorID zu dem die folgenden Daten gehören (0 bis 7)**
	<int>	WindDigital	**Windwert (Details siehe Kap.5.1.3)**
	<int>	RainDigital	**Regenwert (Details siehe Kap.5.1.3)**
	<int>	TempDigital	**Temperturbereich (Siehe Kap.5.1.3)**
	<int>	LightDigital	**Helligkeit (Siehe Kap.5.1.3)**
	<int>	SensorStatus	**0 = Noch keine gültigen Sensordaten verfügbar** **1 = Sensordaten verfügbar** **2 = Sensordaten verfügbar, der Sensor meldet eine schwache Batterie.** **3 = Die Kommunikation zum Sensor ist abgebrochen – „Sensorverlust“.** **4 = Testmodus im Sensor aktiv** **5 = Servicemodus im Sensor aktiv**
	<int>	TempAnalog	**Analoger Temperaturwert (Siehe Kap.5.1.3)**
	<int>	WindAnalog	**Analoger Windwert (Siehe Kap.5.1.3)**
	<int>	Sun1Analog	**Analoger Sonnenwert (Siehe Kap.5.1.3)**
	<int>	DayLightAnalog	**Analoger Tageslichtwert (Siehe Kap.5.1.3)**
	<int>	Sun2Analog	**Analoger Wert des zweiten Sonnensensors (Siehe Kap.5.1.3)**
	<int>	Sun3Analog	**Analoger Wert des dritten Sonnensensors (Siehe Kap.5.1.3)**

5.8 selve.GW.sensor.setLabel

Methode:	**selve.GW.sensor.setLabel**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.setLabel“ kann der Name des Sensors im Gateway geändert werden. Dieser Name ist eine Bezeichnung des Sensors, der nur im Gateway gespeichert wird. **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**		
Parameter: <methodCall>	**<int>**	SensorID	**Nummer des Sensors (0 bis 7) der den Namen erhalten soll.**
	<string>	Name	**Neuer Name des Sensors (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.setLabel**
	<int>	Return	**0 = Der Name konnte nicht übernommen werden.** **1 = Der Name konnte erfolgreich geändert werden.**

5.9 selve.GW.sensor.delete

Methode:	**selve.GW.sensor.delete**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.delete“ kann ein Sensor aus dem Gateway gelöscht werden.		
Parameter: <methodCall>	**<int>**	SensorID	**Nummer des Sensors (0 bis 7) der gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.delete**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Der Sensor konnte erfolgreich gelöscht werden.**

5.10 selve.GW.sensor.writeManual

Methode:	**selve.GW.sensor.writeManual**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sensor.writeManual“ kann ein Sensor manuell im Gateway hinzugefügt werden. Diese Methode kann im Falle eines Backups genutzt werden. Folgende Randbedinungen sind dabei zu erfüllen, damit das manuelle Einrichten des Sensors durchgeführt wird: 1. Die gewählte SensorID darf nicht bereits vom Gateway genutzt werden. Nur bei nicht genutzter SensorID erlaubt das Gateway ein manuelles Einrichten. 2. Die Funkadresse des Sensors darf im Gateway noch nicht hinter einer anderen SensorID vorliegen. **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**		
Parameter: <methodCall>	**<int>**	SensorID	**SensorID, die für das manuelle Einrichten genutzt wird (0 bis 7).**
	<int>	Funkadresse	**Funkadresse des Sensors (0xD80000 bis 0xEFFFFF)**
	<string>	Name	**Name des Sensors (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sensor.writeManual**
	<int>	Return	**0 = Das manuelle Einrichten des Sensors ist fehlgeschlagen.** **1 = Der Sensor konnte erfolgreich im Gateway eingerichtet werden.**

6 selve.GW.senSim – Sensorsimulation

Die Methodenklasse **selve.GW.senSim** beinhaltet alle Methoden, der internen Sensorsimulation. Diese erlaubt es vorhandene Sensoren aus einem Fremdsystem über die Anwendung in das **commeo** Funksystem zu integrieren.

Für die Simulation stehen bis zu 8 **commeo** Sensorsimulationen bereit, die individuell über das Gateway simuliert werden können. Jeder Sensor bekommt eine feste SenSimID (zwischen 0 und 7) zugewiesen, mit der die entsprechenden Daten abgerufen und verwaltet werden können.

HINWEIS **Ein commeo Aktor unterstützt i.d.R. immer nur eine Sensorik. Wird eine zweite Sensorik in einen Aktor eingelernt, so wird die vorherige automatisch überschrieben. Dieses Punkt ist bei der Zuordnung und Vergabe der hier beschriebenen Sensorsimulationen zu berücksichtigen.**

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.senSim.store`	Die Sensorsimulation wird in den ausgewählten Aktor eingelernt.
`selve.GW.senSim.delete`	Die Sensorsimulation wird aus dem ausgewählten Aktor ausgelernt.
`selve.GW.senSim.getConfig`	Lesen der Konfiguration der ausgewählten Sensorsimulation.
`selve.GW.senSim.setConfig`	Konfiguration der ausgewählten Sensorsimulation.
`selve.GW.senSim.setLabel`	Benennen der Sensorsimulation.
`selve.GW.senSim.setValues`	Setzen der entsprechenden Sensorwerte.
`selve.GW.senSim.getValues`	Lesen der aktuell vorliegenden Sensorwerte.
`selve.GW.senSim.getIDs`	Lesen einer Maske, die zeigt welche SenSimIDs aktiv genutzt werden.
`selve.GW.senSim.factory`	Löscht eine Sensorsimulation und setzt die SenSimID auf Werkszustand zurück.
`selve.GW.senSim.drive`	Sendet ein Fahrbefehl zur visuellen Kontrolle.
`selve.GW.senSim.setTest`	Setzen des Testmodus.
`selve.GW.senSim.getTest`	Lesen des aktuellen Status des Testmodus.

Tabelle 7. *Übersicht der Methodenklasse „selve.GW.senSim“*

6.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen zur Simulation von **commeo** Sensoren.

6.1.1 Einlernen einer Sensorsimulation

HINWEIS **Aus Sicherheitsgründen kann eine Sensorsimulation nur in Aktore eingelernt werden, die dem Gateway zum Zeitpunkt des Einlernvorgangs vorliegen.**

Wird im Nachhinein ein Aktor aus dem Gateway entfernt, so bleibt die Sensorsimulation jedoch weiterhin im Aktor eingelernt!

Das Einlernen einer Sensorsimulation erfolgt über die Methode **selve.GW.senSim.store**. Für den Aufruf muss der ausgewählte Aktor in Funkreichweite sein, da an dieser Stelle ein automatisiertes Einlernen im Aktor erfolgt.

Der Einlernprozess ist an dieser Stelle beendet, erst wenn die ersten gültigen Sensordaten mit der Methode **selve.GW.senSim.setValues** dem Gateway vorliegen, wird die interne Sensorsimulation gestartet.

HINWEIS **Nach einem Neustart des Gateways muss die Anwendung dafür sorgen, dass gültige Sensordaten dem Gateway mitgeteilt werden. Ansonsten kann es zu einem Sensorverlust in den entsprechenden Aktoren kommen.**

6.1.2 Auslernen einer Sensorsimulation

Das Auslernen einer Sensorsimulation erfolgt analog zum Einlernprozess. Über die Methode **selve.GW.senSim.delete** wird die Sensorsimulation aus dem ausgewählten Aktor ausgelernt. Für den Aufruf muss der ausgewählte Aktor in Funkreichweite sein.

6.1.3 Löschen einer Sensorsimulation

HINWEIS **Bevor eine Sensorsimulation zurückgesetzt wird, sollte sichergestellt werden, dass die Sensorsimulation aus allen Aktoren ausgelernt ist. Ansonsten kann es zu einem Sensorverlust in den entsprechenden Aktoren kommen.**

Das Löschen einer Sensorsimulation erfolgt über die Methode **selve.GW.senSim.factory**. Mit dem Aufruf wird die ausgewählte Sensorsimulation auf Werkszustand zurückgesetzt und ist entsprechend deaktiviert.

6.2 selve.GW.senSim.store

Methode:	**selve.GW.senSim.store**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.store“ wird die ausgewählte Sensorsimulation als Sensorik in den entsprechenden Aktor eingelernt. **HINWEIS Eine bislang eingelernte Sensorik im Aktor wird i.d.R. überschrieben, wenn der Aktor nur eine Sensorik gleichzeitig unterstützt.**		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	AktorID	**Nummer des Aktors (0 bis 63) in dem die Sensorsimulation eingelernt werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.store**
	<int>	Return	**0 = Ein Fehler beim Einlernen der Sensorsimulation ist aufgetreten.** **1 = Die Sensorsimulation konnte erfolgreich eingelernt werden.**

6.3 selve.GW.senSim.delete

Methode:	**selve.GW.senSim.delete**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.delete“ wird die ausgewählte Sensorsimulation aus dem entsprechenden Aktor ausgelernt.		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	AktorID	**Nummer des Aktors (0 bis 63) in dem die Sensorsimulation ausgelernt werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.delete**
	<int>	Return	**0 = Ein Fehler beim Auslernen der Sensorsimulation ist aufgetreten.** **1 = Die Sensorsimulation konnte erfolgreich ausgelernt werden.**

6.4 selve.GW.senSim.getConfig

Methode:	**selve.GW.senSim.getConfig**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.getConfig“ kann die aktuell vorliegende Konfiguration der ausgewählten Sensorsimulation gelesen werden.		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.getConfig**
	<int>	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	Activity	**0 = Simulation ist deaktiviert.** **1 = Simulation ist aktiviert.**
	<string>	Name	**Name der Sensorsimulation (Maximal 23 Byte)**

6.5 selve.GW.senSim.setConfig

Methode:	**selve.GW.senSim.setConfig**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.setConfig“ erhält die ausgewählte Sensorsimulation ihre grundlegende Konfiguration.		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	Activity	**0 = Simulation ist deaktiviert.** **1 = Simulation ist aktiviert.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.setConfig**
	<int>	Return	**0 = Die Konfiguration konnte nicht übernommen werden.** **1 = Die Konfiguration wurde erfolgreich übernommen.**

6.6 selve.GW.senSim.setLabel

Methode:	**selve.GW.senSim.setLabel**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.setLabel“ kann der Name der Sensorsimulation im Gateway geändert werden. Dieser Name ist eine Bezeichnung, die nur im Gateway gespeichert wird.		
	HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<string>	Name	**Name der Simulation(Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.setLabel**
	<int>	Return	**0 = Der Name konnte nicht übernommen werden.**
			1 = Der Name konnte erfolgreich geändert werden.

6.7 selve.GW.senSim.setValues

Methode:	**selve.GW.senSim.setValues**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.setValues“ werden die Sensordaten der ausgewählten Sensorsimulation aktualisiert. Das Gateway stellt den Aktoren diese Daten automatisch zur Verfügung.		
	HINWEIS Die Normierung der einzelnen Sensordaten ist identisch zu den in Kapitel 5.1.3 beschriebenen Variablen.		
	HINWEIS Erst wenn dem Gateway die ersten Sensordaten mitgeteilt wurden, stellt die Sensorsimulation die Daten den Aktoren zur Verfügung. Demnach müssen nach jedem Neustart die Sensordaten im Gateway aktualisiert werden!		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	WindDigital	**Windwert (Details siehe Kap.5.1.3)**
	<int>	RainDigital	**Regenwert (Details siehe Kap.5.1.3)**
	<int>	TempDigital	**Temperturbereich (Siehe Kap.5.1.3)**
	<int>	LightDigital	**Helligkeit (Siehe Kap.5.1.3)**
	<int>	TempAnalog	**Analoger Temperaturwert (Siehe Kap.5.1.3)**
	<int>	WindAnalog	**Analoger Windwert (Siehe Kap.5.1.3)**
	<int>	Sun1Analog	**Analoger Sonnenwert (Siehe Kap.5.1.3)**
	<int>	DayLightAnalog	**Analoger Tageslichtwert (Siehe Kap.5.1.3)**
	<int>	Sun2Analog	**Analoger Wert des zweiten Sonnensensors (Siehe Kap.5.1.3)**
	<int>	Sun3Analog	**Analoger Wert des dritten Sonnensensors (Siehe Kap.5.1.3)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.setValues**
	<int>	Return	**0 = Daten konnten nicht übernommen werden.**
			1 = Daten wurden übernommen.

6.8 selve.GW.senSim.getValues

Methode:	**selve.GW.senSim.getValues**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.getValues“ werden die aktuellen Sensordaten der ausgewählten Sensorsimulation gelesen. **HINWEIS Die Normierung der einzelnen Sensordaten ist identisch zu den in Kapitel 5.1.3 beschriebenen.**		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.getValues**
	<int>	SenSimID	**Nummer der ausgewählten Sensorsimulation (0 bis 7).**
	<int>	WindDigital	**Windwert (Details siehe Kap.5.1.3)**
	<int>	RainDigital	**Regenwert (Details siehe Kap.5.1.3)**
	<int>	TempDigital	**Temperturbereich (Siehe Kap.5.1.3)**
	<int>	LightDigital	**Helligkeit (Siehe Kap.5.1.3)**
	<int>	TempAnalog	**Analoger Temperaturwert (Siehe Kap.5.1.3)**
	<int>	WindAnalog	**Analoger Windwert (Siehe Kap.5.1.3)**
	<int>	Sun1Analog	**Analoger Sonnenwert (Siehe Kap.5.1.3)**
	<int>	DayLightAnalog	**Analoger Tageslichtwert (Siehe Kap.5.1.3)**
	<int>	Sun2Analog	**Analoger Wert des zweiten Sonnensensors (Siehe Kap.5.1.3)**
	<int>	Sun3Analog	**Analoger Wert des dritten Sonnensensors (Siehe Kap.5.1.3)**

6.9 selve.GW.senSim.getIDs

Methode:	**selve.GW.senSim.getIDs**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.getIDs“ kann eine Maske gelesen werden, die zeigt welche SenSimIDs im Gateway aktuell in Benutzung sind. Diese Methode ermöglicht einen optimierten Zugriff. Werden z.B. während der Initialisierungphase einer Anwendung alle Daten erfasst, so kann durch diese Maske das Lesen der notwendigen Daten beschleunigt werden. Die Maske der SenSimIDs ist genau 1 Byte lang. Jedes Bit repräsentiert eine SenSimID. Ist das Bit auf 1 gesetzt, so handelt es sich bei dieser SenSimID um eine genutzte Sensorsimulation. Eine 0 zeigt, dass diese SenSimID nicht genutzt wird. Byte 0.0 SenSimID 0 ... Byte 0.7 SenSimID 7		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.getIDs**
	<base64>	Maske	**Maske der genutzten SenSimIDs**

6.10 selve.GW.senSim.factory

Methode:	**selve.GW.senSim.factory**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.factory“ kann eine Sensorsimulation komplett gelöscht werden und auf Werkszustand gesetzt werden.		
	HINWEIS Bevor die Sensorsimulation zurückgesetzt wird, sollte die Sensorsimulation aus allen Aktoren ausgelernt werden, damit ein möglicher Sensorverlust ausgeschlossen werden kann.		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der Sensorsimulation (0 bis 7) die gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.factory**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Die Sensorsimulation konnte erfolgreich gelöscht werden.**

6.11 selve.GW.senSim.drive

Methode:	**selve.GW.senSim.drive**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.drive“ kann eingeschränkt ein Fahrbefehl gesendet werden.		
	HINWEIS Die hier genutzten Fahrbefehle sind so ausgelegt, dass die Endlagen am Aktor eingestellt sein müssen!		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der Sensorsimulation (0 bis 7) die gelöscht werden soll.**
	<int>	Kommando	**Das Kommando, welches ausgeführt werden soll:** **0 = Stop** **1 = DriveUp** **2 = DriveDown** **3 = DrivePos1** **5 = DrivePos2**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.drive**
	<int>	Return	**0 = Ein Fehler ist aufgetreten.** **1 = Der Befehl wurde gesendet.**

6.12 selve.GW.senSim.setTest

Methode:	**selve.GW.senSim.setTest**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.senSim.setTest“ kann der Testmodus innerhalb einer Sensorsimulation gestartet, bzw. beendet werden. Ist der Testmodus aktiv, so wird die interne Sensorzeit um Faktor 20 beschleunigt (D.h. aus 2 Minuten werden 6 Sekunden). **HINWEIS Der Testmodus wird intern nach 5 Minuten automatisch beendet.**		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der Sensorsimulation (0 bis 7).**
	<int>	TestMode	**0 = Testmodus ausgeschaltet** **1 = Testmodus aktiviert**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.setTest**
	<int>	Return	**0 = Der Befehl wurde nicht durchgeführt.** **1 = Der Befehl wurde durchgeführt.**

6.13 selve.GW.senSim.getTest

Methode:	**selve.GW.senSim.getTest**		
Beschreibung:	Die Methode „selve.GW.senSim.getTest“ ermöglicht das Lesen der Testmodus Aktivität. Ist der Testmodus aktiv, so wird die interne Sensorzeit um Faktor 20 beschleunigt (D.h. aus 2 Minuten werden 6 Sekunden). **HINWEIS Der Testmodus wird intern nach 5 Minuten automatisch beendet.**		
Parameter: <methodCall>	**<int>**	SenSimID	**Nummer der Sensorsimulation (0 bis 7).**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.senSim.getTest**
	<int>	SenSimID	**Nummer der Sensorsimulation (0 bis 7).**
	<int>	TestMode	**0 = Testmodus ist ausgeschaltet** **1 = Testmodus ist aktiviert**

7 selve.GW.sender – Senderverwaltung

Die Methodenklasse **selve.GW.sender** beinhaltet alle Methoden, die für das Einlernen, und der Verwaltung von **commeo** Sendern notwendig sind.

Mit dieser Klasse ist es möglich Kommandos von **commeo** Sendern der Anwendung zur Verfügung zu stellen. Hierzu generiert das Gateway Ereignisse, die der Anwendung bei Tastendruck automatisch gesendet werden. So können z.B. über die **commeo** Sender externe Fremdsysteme angesteuert werden.

Es können bis zu 63 Kanäle von **commeo** Sendern in einem Gateway eingelernt und verwaltet werden. Jeder Sender bekommt eine feste SenderID (zwischen 0 und 62) zugewiesen. Mit dieser SenderID können die entsprechenden Daten abgerufen und verwaltet werden.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.sender.teachStart`	Startet einen Einlernprozess eines **commeo** Senders.
`selve.GW.sender.teachStop`	Stoppt einen laufenden Sender-Lernprozess.
`selve.GW.sender.teachResult`	Liefert den aktuellen Fortschritt des Sender-Lernprozesses.
`selve.GW.sender.getIDs`	Lesen einer Bitmaske, die beschreibt welche SenderIDs aktuell genutzt werden.
`selve.GW.sender.getInfo`	Auslesen der Senderinformationen.
`selve.GW.sender.getValues`	Lesen der aktuellen Werte des Senders.
`selve.GW.sender.setLabel`	Setzen eines Labels für den Sender.
`selve.GW.sender.delete`	Löscht einen Sender aus der Liste.
`selve.GW.sender.writeManual`	Manuelles Schreiben eines Senders in die Liste

Tabelle 8. *Übersicht der Methodenklasse „selve.GW.sender“*

7.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen um Sender im Gateway hinzuzufügen, bzw. zu löschen.

HINWEIS **Um hier eine korrekte Verwaltung sicherzustellen ist das Einlernen und Auslernen im Gateway stets ausgeschaltet und muss explizit gestartet werden.**

7.1.1 Einlernen von Sender

Das Einlernen eines Senders in das Gateway erfolgt analog zum Einlernprozess zwischen einem **commeo** Sender und **commeo** Aktor.

Das Einlernen eines Senders erfolgt in den folgenden Schritten:

- Um einen Sender im Gateway einlernen zu können muss mit der Methode **selve.GW.sender.teachStart** das Gateway in den Einlernprozess gebracht werden. Sind alle SenderIDs vergeben, so wird dies mit einem Fehler quittiert. Der Einlernprozess ist nach Aufruf der Methode für 4 Minuten aktiv und schaltet sich, wenn kein Sender eingelernt wird, automatisch nach dieser Zeit ab. Die Methode **selve.GW.sender.teachResult** liefert an dieser Stelle zyklisch einen Status, der die noch verbleibende Zeit mitteilt.
- Im zweiten Schritt wird über die SELECT Taste des entsprechenden **commeo** Senders ein Suchlauf gestartet, der nun die freigeschaltete, freie SenderID des Gateways findet und in seiner Liste aufnehmen kann.
- Mit der SELECT Taste des **commeo** Senders wird das Gateway selektiert. Das selektieren des Gateways, sowie die an dieser Stelle betätigten Fahrkommandos werden über die Methode **selve.GW.sender.teachResult** als Ereignis der Anwendung für eine visuelle Anzeige zur Verfügung gestellt.
- Mit dem Betätigen der PROG Taste des **commeo** Senders wird die Verbindung zwischen den Sender und dem Gateway hergestellt und der Einlernprozess ist beendet.
- Über die Methode **selve.GW.sender.teachResult** wird das erfolgreiche Einlernen des Senders automatisch der Anwendung mitgeteilt. Gleichzeitig wird der Einlernprozess an dieser Stelle beendet. Für das Einlernen mehrere Sender muss der Einlernprozess wiederholt werden.

7.1.2 Auslernen von Sender

Das Auslernen eines Senders aus dem Gateway erfolgt analog dem Auslernprozess zwischen einem **commeo** Sender und einem **commeo** Aktor.

Das Auslernen eines Senders erfolgt in den folgenden Schritten:

- Über die SELECT Taste des entsprechend, eingelernten **commeo** Sendern wird der Suchlauf gestartet.
- Mit der SELECT Taste des **commeo** Senders wird der Kanal des Gateways selektiert. Für eine visuelle Anzeige dient das entsprechende ‚Select' Ereignis.
- Mit dem Betätigen der PROG Taste des **commeo** Senders wird die Verbindung zwischen den Sender und dem Gateway aufgehoben und der Anwendung mit einem letzten ‚Delete' Ereignis mitgeteilt.

  Im Gateway wird die Senderadresse gelöscht und die Anwendung kann durch das Ereignis ebenfalls ihre Daten aktualisieren.

7.1.3 Löschen von Sender

Das Löschen von Sender aus dem Gateway ist jederzeit über die Methode **selve.GW.sender.delete** erreichbar. Wird diese Methode aufgerufen, so wird die SenderID im Gateway auf Werkszustand gesetzt.

HINWEIS **Das Löschen von SenderIDs sollte nur in Ausnahmefällen, bei z.B. Verlust des Senders, durchgeführt werden. Die noch eingelernte SenderID muss im zweiten Schritt manuell aus dem entsprechenden commeo Sender gelöscht werden.**

7.1.4 Sendereereignisse

In diesem Abschnitt werden die einzelnen Ereignisse, die der Anwendung von einem **commeo** Sender gemeldet werden, beschrieben. Über die Methode **selve.GW.sender.getValues** kann im Nachhinein das letzte Ereignis des Senders nochmals gelesen werden.

Ereignis	Wert	Beschreibung
Unknown	0	Es liegt kein Ereignis vor (Zustand nach Start)
DriveUp	1	Der Sender meldet, dass die Auffahrttaste betätigt wurde.
DriveDown	2	Der Sender meldet, dass die Abfahrtstaste betätigt wurde.
Stop	3	Der Sender meldet, dass die Stop Taste betätigt wurde.
ZwPos1	4	Der Sender meldet den Aufruf der Zwischenposition 1 (Verschattungsposition).
ZwPos2	5	Der Sender meldet den Aufruf der Zwischenposition 2 (Lüftungsposition).
SavePos1	6	Der Sender meldet das Speichern der Zwischenposition 1.
SavePos2	7	Der Sender meldet das Speichern der Zwischenposition 2.
Auto	8	Der Sender meldet dass der Schiebeschalter auf AUTO gesetzt wurde.
Man	9	Der Sender meldet dass der Schiebeschalter auf MAN gesetzt wurde.
Name	10	Der Sender meldet, dass der Name des Senders von z.B. einem Commeo MultiSend verändert wurde.
KeyRelease	11	Der Sender meldet das Loslassen einer Fahrtaste. (Dieses Ereignis wird nur im Lernprozess generiert).
Select	12	Der Sender meldet das Selektieren des Gateways.
Delete	13	Die Verbindung zwischen dem Sender und dem Gateway wurde gelöscht und steht nun für weitere Ereignisse nicht mehr zur Verfügung.

Tabelle 9. *Übersicht der definierten Sendereereignisse.*

7.2 selve.GW.sender.teachStart

Methode:	**selve.GW.sender.teachStart**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.teachStart“ wird das Einlernen eines neuen Senders für 4 Minuten im Gateway aktiviert. Sind bereits alle 63 SenderIDs vergeben, so wird der Start eines neuen Senders an dieser Stelle mit einem Fehler quittiert. Ist bereits ein Einlernprozess gestartet, wird dieser gestoppt und ein neuer Einlernprozess gestartet.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.teachStart**
	<int>	Return	**0 = Der Einlernprozess konnte nicht gestartet werden.** **1 = Der Einlernprozess ist gestartet.**

7.3 selve.GW.sender.teachStop

Methode:	**selve.GW.sender.teachStop**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.teachStop“ wird der aktuelle Sender-Lernmodus gestoppt und die Einlernbereitschaft im Gateway deaktiviert.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.teachStop**
	<int>	Return	**0 = Der Lernprozess konnte nicht gestoppt werden.** **1 = Der Lernprozess ist gestoppt worden.**

7.4 selve.GW.sender.teachResult

Methode:	**selve.GW.sender.teachResult**
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt und meldet zyklisch den Fortschritt des Sender-Lernprozesses.** **Alternativ kann die gleiche Methode auch dazu genutzt werden, um von der Anwendung manuell einen aktuellen Status des Lernprozesses anzufragen. In diesem Fall sind bei der Anfrage an das Gateway keine Parameter erforderlich und das Gateway antwortet entsprechend der üblichen Form mit dem Methodennamen und den unten beschrieben Parametern.** Mit Hilfe der Methode „selve.GW.sender.teachResult“ wird der Anwendung der aktuelle Status eines laufenden Sender-Lernprozesses mitgeteilt. Folgende Zustände werden über den Status mitgeteilt: **Idle** Steht der Zustand auf diesen Wert, so ist kein Sender-Lernprozess gestartet. **TeachIn** In diesem Zustand ist ein Sender-Einlernprozess gestartet und ein Sender kann eingelernt werden. Ist die interne Zeit abgelaufen, so wird der Prozess automatisch beendet und der Prozess wechselt zurück in den Idle Zustand. **TeachInSuccess** Dieser Zustand signalisiert der Anwendung das ein Sender erfolgreich eingelernt wurde. Die mitgelieferte SenderID zeigt an dieser Stelle den neu eingelernten Sender.
Parameter: <methodCall>	**<int>** Status — **0 = Idle** **1 = TeachIn** **2 = TeachInSuccess** **<int>** TimeLeft — **Angabe in Sekunden, wie lange der Lernprozess noch aktiv ist.** **<int>** SenderID — **-1 = Keine gültigen Sender-informationen verfügbar.** **0 bis 62 = SenderID des eingelernten Senders.** **<string>** Name — **Name der entsprechenden SenderID (Maximal 23 Byte)** **<int>** Event — **Ereignisse nach der Definition aus Kapt.7.1.4. An dieser Stelle werden folgende Ereignisse gemeldet:** **0 – Unknown** **1 – DriveUp** **2 - DriveDown** **3 - Stop** **11 - KeyRelease** **12 - Select**
Parameter: <methodResponse>	---

7.5 selve.GW.sender.getIDs

Methode:	**selve.GW.sender.getIDs**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.getIDs“ kann eine Maske gelesen werden, die zeigt welche SenderIDs im Gateway aktuell in Benutzung sind und ermöglicht einen optimierten Zugriff. Die Maske der SenderIDs ist genau 8 Bytes lang. Jedes Bit repräsentiert eine SenderID. Ist das Bit auf 1 gesetzt, so handelt es sich bei diese SenderID um einen genutzten Sender. Eine 0 zeigt, dass diese SenderID nicht genutzt wird und bei z.B. einem kompletten Download nicht relevant ist. Byte 0.0 SenderID 0 ... Byte 0.7 SenderID 7 Byte 1.0 SenderID 8 ... Byte 7.6 SenderID 62		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.getIDs**
	<base64>	Maske	**Maske der genutzten SenderIDs**

7.6 selve.GW.sender.getInfo

Methode:	**selve.GW.sender.getInfo**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.getInfo“ werden die aktuell vorliegenden Informationen der ausgewählten SenderID gelesen. Zu dem aktuellen Namen wird an dieser Stelle auch die commeo Funkinformationen des Senders und der aktuelle Speicherzustand mitgeteilt. Diese Daten sind für ein mögliches Backup der Anwendung relevant. **HINWEIS SenderIDs mit der Funkadresse 0x000000 sind als unbenutzt zu kennzeichnen.**		
Parameter: <methodCall>	**<int>**	SenderID	**Nummer des Senders (0 bis 62) von dem die Daten gelesen werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.getInfo**
	<int>	SenderID	**SenderID zu dem die folgenden Daten gehören (0 bis 62)**
	<int>	Funkadresse	**Funkadresse des Senders (0x800000 bis 0xBFFFFF)**
	<int>	Funkkanal	**Zugehöriger Funkkanal des Senders (0 bis 63)**
	<int>	FunkResetCount	**Resetcounter des Senders (0 bis 255).**
	<string>	Name	**Aktueller Name des Senders (UTF-8 formatiert).**

7.7 selve.GW.sender.getValues

Methode:	**selve.GW.sender.getValues**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.getValues“ werden die aktuell vorliegenden Senderdaten des ausgewählten Senders gelesen.		
Parameter: <methodCall>	**<int>**	SenderID	**Nummer des Senders (0 bis 62) von dem die Daten gelesen werden sollen.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.getValues**
	<int>	SenderID	**SenderID zu dem die folgenden Daten gehören (0 bis 62)**
	<int>	LastEvent	**Letztes, vorliegendes Ereignis des Sender (Siehe Kap.7.1.4)**

7.8 selve.GW.sender.setLabel

Methode:	**selve.GW.sender.setLabel**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.setLabel“ kann der Name des Senders im Gateway geändert werden. **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**		
Parameter: <methodCall>	**<int>**	SenderID	**Nummer des Senders (0 bis 62) der den Namen erhalten soll.**
	<string>	Name	**Neuer Name des Senders (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.setLabel**
	<int>	Return	**0 = Der Name konnte nicht übernommen werden.** **1 = Der Name konnte erfolgreich geändert werden.**

7.9 selve.GW.sender.delete

Methode:	**selve.GW.sender.delete**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.delete“ kann ein Sender aus dem Gateway gelöscht werden. **HINWEIS Das Löschen einer SenderID im Gateway sorgt dafür, dass in dem entsprechenden commeo Sender ein manuelles Löschen erfolgen muss!**		
Parameter: <methodCall>	**<int>**	SenderID	**Nummer des Senders (0 bis 62) der gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.delete**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Der Sender konnte erfolgreich gelöscht werden.**

7.10 selve.GW.sender.writeManual

Methode:	**selve.GW.sender.writeManual**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.sender.writeManual“ kann ein Sender manuell im Gateway hinzugefügt werden. Diese Methode kann im Falle eines Backups genutzt werden. Folgende Randbedingungen sind dabei zu erfüllen, damit das manuelle Einrichten des Senders durchgeführt wird: 1. Die gewählte SenderID darf nicht bereits vom Gateway genutzt werden. Nur bei nicht genutzter SenderID erlaubt das Gateway ein manuelles Einrichten. 2. Die Funkadresse des Senders darf im Gateway noch nicht hinter einer anderen SenderID vorliegen. **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**		
Parameter: <methodCall>	**<int>**	SenderID	**SenderID zu dem die folgenden Daten gehören (0 bis 62)**
	<int>	Funkadresse	**Funkadresse des Senders (0x800000 bis 0xBFFFFF)**
	<int>	Funkkanal	**Zugehöriger Funkkanal des Senders (0 bis 63)**
	<int>	FunkResetCount	**Resetcounter des Senders (0 bis 255).**
	<string>	Name	**Name des Senders (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.sender.writeManual**
	<int>	Return	**0 = Das manuelle Einrichten des Senders ist fehlgeschlagen.** **1 = Der Sender konnte erfolgreich im Gateway eingerichtet werden.**

8 selve.GW.group – commeo Gruppenverwaltung

Die Methodenklasse **selve.GW.group** beinhaltet alle Methoden, die es ermöglichen vordefinierte, immer wieder verwendete, Gruppen von **commeo** Aktore zu verwalten, so dass die Anwendung einen schnelleren Zugriff auf Gruppenkommandos erhält.

Es werden bis zu 32 individuell zu konfigurierende Gruppen in einem Gateway verwaltet. Jede Gruppe bekommt eine feste GruppenID (zwischen 0 und 31) zugewiesen. Mit dieser GruppenID können die entsprechenden Daten abgerufen und verwaltet werden.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
selve.GW.group.read	Lesen einer Gruppenkonfiguration.
selve.GW.group.write	Schreiben einer Gruppenkonfiguration.
selve.GW.group.getIDs	Lesen einer Bitmaske, die beschreibt welche GruppenIDs aktuell mit Aktoren belegt sind.
selve.GW.group.delete	Löschen einer Gruppenkonfiguration.

Tabelle 10. *Übersicht der Methodenklasse „selve.GW.group“*

8.1 selve.GW.group.read

Methode:	**selve.GW.group.read**
Beschreibung:	Mit Hilfe der Methode „selve.GW.group.read“ kann die aktuelle Konfiguration einer Gruppe gelesen werden. Die Konfiguration beinhaltet eine Bitmaske, die beschreibt welche AktorIDs der Gruppe zugehören. Weiterhin kann einer Gruppe einen Namen gegeben werden, die eine bessere Darstellung ermöglicht. Die Maske der Gruppe ist genau 8 Byte lang. Jedes Bit repräsentiert eine AktorID. Ist das Bit auf 1 gesetzt, so ist dieser Aktor Mitglied der Gruppe. Eine 0 zeigt, dass die entsprechende AktorID nicht zur Gruppe gehört. Byte 0.0 AktorID 0 ... Byte 0.7 AktorID 7 Byte 1.0 AktorID 8 ... Byte 7.7 AktorID 63
Parameter: <methodCall>	**<int>** GruppenID **Nummer der ausgewählten Gruppe (0 bis 31), die gelesen werden soll.**
Parameter: <methodResponse>	**<string>** Methodenname **selve.GW.group.read** **<int>** GruppenID **Nummer der Gruppe (0 bis 31), zu der die folgenden Daten gehören.** **<base64>** Maske **Maske der zugehörigen AktorIDs der Gruppe.** **<string>** Name **Name der Gruppe (UTF-8 formatiert).**

8.2 selve.GW.group.write

Methode:	**selve.GW.group.write**
Beschreibung:	Mit Hilfe der Methode „selve.GW.group.write“ kann die aktuelle Konfiguration einer Gruppe aktualisiert werden. Die Konfiguration beinhaltet die Bitmaske der zugehörigen Gruppenmitglieder und einen Namen, der für eine bessere Darstellung genutzt werden kann. Die Maske der Gruppe ist genau 8 Byte lang. Jedes Bit repräsentiert eine AktorID. Ist das Bit auf 1 gesetzt, so ist dieser Aktor Mitglied der Gruppe. Eine 0 zeigt, dass die entsprechende AktorID nicht zur Gruppe gehört. Byte 0.0 AktorID 0 ... Byte 0.7 AktorID 7 Byte 1.0 AktorID 8 ... Byte 7.7 AktorID 63 **HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!**
Parameter: <methodCall>	**<int>** GruppenID **Nummer der ausgewählten Gruppen (0 bis 31), die aktualisiert werden soll.** **<base64>** Maske **Maske der zugehörigen AktorIDs der Gruppe.** **<string>** Name **Name der Gruppe (Maximal 23 Byte).**
Parameter: <methodResponse>	**<string>** Methodenname **selve.GW.group.write** **<int>** Return **0 = Das Schreiben der Konfiguration ist fehlgeschlagen.** **1 = Das Schreiben der Konfiguration war erfolgreich.**

8.3 selve.GW.group.getIDs

Methode:	**selve.GW.group.getIDs**
Beschreibung:	Mit Hilfe der Methode „selve.GW.group.getIDs“ kann eine Maske gelesen werden, die zeigt welche GruppenIDs im Gateway aktuell in Benutzung sind und ermöglicht einen optimierten Zugriff. Die Maske der GruppenIDs ist genau 4 Bytes lang. Jedes Bit repräsentiert eine GruppenID. Ist das Bit auf 1 gesetzt, so handelt es sich bei diese GruppenID um eine genutzte Gruppe. Eine 0 zeigt, dass diese GruppenID nicht genutzt wird und bei z.B. einem kompletten Download nicht relevant ist. Byte 0.0 GruppenID 0 ... Byte 0.7 GruppenID 7 Byte 1.0 GruppenID 8 ... Byte 3.7 GruppenID 31
Parameter: <methodCall>	---
Parameter: <methodResponse>	**<string>** Methodenname **selve.GW.group.getIDs** **<base64>** Maske **Maske der genutzten GruppenIDs.**

8.4 selve.GW.group.delete

Methode:	**selve.GW.group.delete**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.group.delete“ kann ein Gruppe im Gateway gelöscht werden. Das Löschen der Gruppe sorgt dafür, dass alle aktuellen Mitglieder der Gruppe entfernt werden und der Name auf den Werkszustand zurückgesetzt wird.		
Parameter: <methodCall>	**<int>**	GruppenID	**Nummer des Gruppe (0 bis 31) die gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.group.delete**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Die Gruppe konnte erfolgreich gelöscht werden.**

9 selve.GW.command – Bedienung der commeo Aktore

Die Methodenklasse **selve.GW.command** beinhaltet alle Methoden, die es ermöglichen **commeo** Aktore einzeln, in vordefinierte Gruppen oder manuell zur Laufzeit definierten Gruppen zu bedienen.

Die folgende Übersicht zeigt die hier verfügbaren Methoden:

Methode	Kurzbeschreibung
`selve.GW.command.device`	Bedienung eines einzelnen Aktors mit Hilfe der AktorID.
`selve.GW.command.group`	Bedienung einer vordefinierten Gruppe mit Hilfe der GruppenID.
`selve.GW.command.groupMan`	Bedienung einer zur Laufzeit definierten Gruppe mit individueller Maske.
`selve.GW.command.result`	Liefert das Ergebnis des Bedienkommandos.

Tabelle 11. *Übersicht der Methodenklasse „selve.GW.command“*

9.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen, die für die Bedienung der commeo Aktore hilfreich sind.

9.1.1 Ablauf einer Bedienung

Der folgende Ablauf zeigt das korrekte Handling eines Bedienkommandos. Es ermöglicht der Anwendung gleichzeitig eine optimale Darstellung der Aktore, die das Kommando erfolgreich ausgeführt haben und die Kennzeichnung der Aktore, die das Kommando nicht ausgeführt haben.

Weitere Details können aus den entsprechenden Zuständen der Aktore entnommen werden.

Das folgende Ablaufdiagramm zeigt die vollständige Abwicklung eines Bedienkommandos.

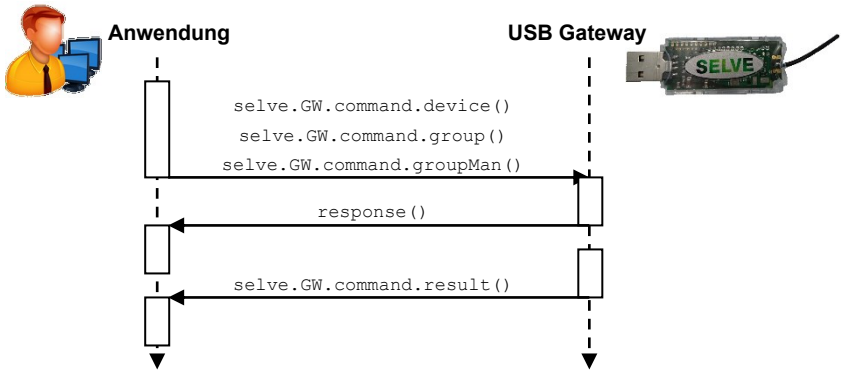

Bild 7. Grafische Darstellung des Ablaufs eines Bedienkommandos.

9.1.2 Bedienkommandos

Die folgende Tabelle beschreibt die im Gateway vorhandenen Kommandos. Diese Kommandos können in den „command“ Befehlen genutzt werden.

Kommando	Wert	Beschreibung
Stop	0	Dieses Kommandos sorgt dafür, das Aktore gestoppt werden. (**commeo** Schaltaktore werden ausgeschaltet).
DriveUp	1	Der Befehl sorgt für eine Aufwärtsfahrt der entsprechenden Aktore. (**commeo** Schaltaktore werden eingeschaltet).
DriveDown	2	Der Befehl sorgt für eine Abwärtsfahrt der entsprechenden Aktore.
DrivePos1	3	Der Befehl sorgt für eine Fahrt in die Zwischenposition 1 (Verschattungsposition).
SavePos1	4	Der Befehl speichert die aktuell eingestellte Position als Zwischenposition 1 ab.
DrivePos2	5	Der Befehl sorgt für eine Fahrt in die Zwischenposition 2 (Lüftungsposition).
SavePos2	6	Der Befehl speichert die aktuell eingestellte Position als Zwischenposition 2 ab.
DrivePos	7	Der Befehl fährt eine, im Parameter vorgegebene Position. Die Position ist wie folgt definiert: 0 = Obere Endlage eines Motors ... 65535 = Untere Endlage eines Motors
StepUp	8	Der Befehl führt für z.B. eine Lammellenverstellung einen Aufwärtsstepp durch. Der übergebene Parameter (0 bis 65535) ermöglicht eine **relative,** anwendungsspezifische Winkelgradeinstellung: 1 = 1° ... 360 = 360° Bei Dimmwerten wird die Helligkeit zum eingestellten Lichtwert hinzuaddiert.
StepDown	9	Der Befehl führt für z.B. eine Lammellenverstellung einen Abwärtsstepp durch. Der übergebene Parameter (0 bis 65535) ermöglicht eine **relative,** anwendungsspezifische Winkelgradeinstellung: 1 = 1° ... 360 = 360° Bei Dimmwerten wird die Helligkeit zum eingestellten Lichtwert angezogen.
AutoOn	10	Der Befehl schaltet die Automatik in den entsprechenden Aktore ein.
AutoOff	11	Der Befehl schaltet die Automatik in den entsprechenden Aktore ab, so dass der manuelle Modus in den Aktoren aktiviert wird.

Tabelle 12. *Übersicht der definierten Bedienkommandos.*

9.1.3 Typen der Bedienkommandos

Bei dem Aufruf der Bedienkommandos wird zusätzlich ein Kommandotyp übergeben, der die Art des Kommandos beschreibt. Folgende Typen sind definiert:

Typ	Wert	Beschreibung
TypeForced	0	Dieser Typ zwingt einen Aktor das Kommando durchzuführen, auch wenn er z.B. keine Endlagen eingestellt hat, oder in einem Alarmzustand steht. **HINWEIS Dieser Typ sollte in der normalen Bedienung nicht verwendet werden. Eine falsche Bedienung kann zur Zerstörung der Anlage führen!**
TypeManual	1	Dieser Typ wird **standardmäßig** für Kommandos genutzt. Es symbolisiert die Nutzung eines übliches Bediengeräts, welcher manuelle Kommandos durchführt.
TypeTime	2	Dieser Typ wird genutzt, wenn die Anwendung automatisch generierte Kommandos sendet. Hierzu gehören Kommandos von Sensoren, Schaltuhren, etc. ist die Automatik im Aktor ausgeschaltet, so werden diese Kommandos nicht durchgeführt.
TypeGlass	3	Dieser Typ signalisiert dem Aktor, dass der Befehl von einem Glasbruchsensor kommt, so dass der Aktor entsprechend seiner Funktion reagieren kann.

Tabelle 13. *Übersicht der definierten Typen der Bedienkommandos.*

9.2 selve.GW.command.device

Methode:	**selve.GW.command.device**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.command.device“ kann ein einzelner, eingelernter Aktor direkt bedient werden.		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) der bedient werden soll.**
	<int>	Kommando	**Das Kommando, welches ausgeführt werden soll (Details siehe Kap.9.1.2)**
	<int>	Typ	**Der erweiterte Typ des Kommandos (Details siehe Kap.9.1.3)**
	<int>	Parameter	**Ein zusätzlicher Parameter, der bei den entsprechenden Kommandos benötigt wird. Bei Kommandos ohne Parameter, wird der Inhalt intern nicht berücksichtigt und kann auf 0 gesetzt werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.command.device**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.**
			1 = Das Kommando wird ausgeführt, das Ergebnis des Kommandos wird nach der Durchführung automatisch mitgeteilt.

9.3 selve.GW.command.group

Methode:	**selve.GW.command.group**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.command.group“ werden alle Aktore einer vordefinierten Gruppe bedient.		
Parameter: <methodCall>	**<int>**	GruppenID	**Nummer der Gruppe (0 bis 31) die bedient werden soll.**
	<int>	Kommando	**Das Kommando, welches ausgeführt werden soll (Details siehe Kap.9.1.2)**
	<int>	Typ	**Der erweiterte Typ des Kommandos (Details siehe Kap.9.1.3)**
	<int>	Parameter	**Ein zusätzlicher Parameter, der bei den entsprechenden Kommandos benötigt wird. Bei Kommandos ohne Parameter, wird der Inhalt intern nicht berücksichtigt und kann auf 0 gesetzt werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.command.group**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.**
			1 = Das Kommando wird ausgeführt, das Ergebnis des Kommandos wird nach der Durchführung automatisch mitgeteilt.

9.4 selve.GW.command.groupMan

Methode:	**selve.GW.command.groupMan**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.command.groupMan“ werden alle Aktore einer anwendungsspezifischen Gruppe bedient. Diese Gruppe wird während der Laufzeit durch die Anwendung gebildet. Die Maske der Gruppe ist genau 8 Byte lang. Jedes Bit repräsentiert eine AktorID. Ist das Bit auf 1 gesetzt, so ist dieser Aktor Mitglied dieser Gruppe. Eine 0 zeigt, dass die entsprechende AktorID nicht zur Gruppe gehört. Byte 0.0 AktorID 0 ... Byte 0.7 AktorID 7 Byte 1.0 AktorID 8 ... Byte 7.7 AktorID 63 **HINWEIS Die Gruppenmaske wird intern mit den genutzten AktorIDs abgeglichen. Alle nicht genutzten AktorIDs werden vor dem Aufruf automatisch entfernt!** **Die daraus resultierende Maske wird in der Antwort mitgeliefert.**		
Parameter: <methodCall>	**<int>**	Kommando	**Das Kommando, welches ausgeführt werden soll (Details siehe Kap.9.1.2)**
	<int>	Typ	**Der erweiterte Typ des Kommandos (Details siehe Kap.9.1.3)**
	<base64>	Maske	**Anwendungsspezifische Maske der AktorIDs, die das Kommando durchführen sollen.**
	<int>	Parameter	**Ein zusätzlicher Parameter, der bei den entsprechenden Kommandos benötigt wird. Bei Kommandos ohne Parameter, wird der Inhalt intern nicht berücksichtigt und kann auf 0 gesetzt werden.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.command.groupMan**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.** **1 = Das Kommando wird ausgeführt, das Ergebnis des Kommandos wird nach der Durchführung automatisch mitgeteilt.**
	<base64>	Maske	**Maske der AktorIDs, die das Kommando wirklich erhalten.**

9.5 selve.GW.command.result

Methode:	**selve.GW.command.result**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt!** Mit Hilfe der Methode „selve.GW.command.result“ wird der Anwendung das Ergebnis des aufgerufenden Kommandos mitgeteilt. Für eine schnelle Zuordnung wird im ersten Teil der Antwort das ausgeführte Kommando mitgeliefert. Im zweiten Teil werden zwei Masken der Anwendung übergeben, die Zeigen welche Aktore das Kommando durchgeführt haben und – im Falle eines Fehlers – bei welchen Aktoren ein Fehler bei der Kommandosausführung festegestellt wurde.		
Parameter: <methodCall>	**<int>**	Kommando	**Das Kommando, welches ausgeführt wurde (Details siehe Kap.9.1.2)**
	<int>	Typ	**Der erweiterte Typ des durchgeführten Kommandos (Details siehe Kap.9.1.3)**
	<int>	Eregbnis	**0 = Bei der Kommandoausführung sind Fehler aufgetreten.** **1 = Das Kommando konnte erfolgreich ausgeführt werden.**
	<base64>	MaskSuccess	**Maske der AktorIDs, die erfolgreich das Kommando ausgeführt haben.**
	<base64>	MaskFailed	**Maske der AktorIDs, die bei denen ein Fehler bei der Kommandosausführung festgestellt wurde.**
Parameter: <methodResponse>	---		

10 selve.GW.event – Ereignismanager

Die Methodenklasse **selve.GW.event** beinhaltet die Methoden, die vom Gateway automatisch als Ereignisse generiert werden.

Je nach Konfiguration des Gateways (siehe Kapitel **selve.GW.param**) werden bei vorliegenden Ereignissen die Anwendung automatisch informiert.

Die folgende Übersicht zeigt die hier erzeugten Ereignisse:

Methode	Kurzbeschreibung
`selve.GW.event.device`	Das Ereignis wird generiert, wenn sich Zustände eines Aktors verändert haben.
`selve.GW.event.sensor`	Das Ereignis wird generiert, wenn Sensordaten eines Sensors aktualisiert wurden.
`selve.GW.event.sender`	Das Ereignis wird generiert, wenn ein neues Ereignis von einem Handsender vorliegt.
`selve.GW.event.log`	Das Ereignis wird generiert, wenn interne Ereignisse vorliegen, die als Meldungen in der Anwendung geloggt werden sollen.
`selve.GW.event.dutyCycle`	Das Ereignis wird generiert, wenn sich Werte bei der Funkauslastung verändert haben.

Tabelle 14. *Übersicht der Methodenklasse „selve.GW.event“*

10.1 selve.GW.event.device

Methode:	**selve.GW.event.device**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt, wenn die Funktion in den Gatewayeinstellungen aktiviert ist!** Mit Hilfe der Methode „selve.GW.event.device“ wird der Anwendung Zustandsänderungen von Aktoren mitgeteilt. **HINWEIS Details zu den einzelnen Parametern sind in der Methodenklasse selve.GW.device zu entnehmen.**		
Parameter: <methodCall>	**<int>**	AktorID	**Nummer des Aktors (0 bis 63) zu dem die folgenden Daten gehören.**
	<int>	Status	**0 = Unbekannter Zustand** **1 = Aktor ist gestoppt, bzw. Aktor ist ausgeschaltet** **2 = Aktor fährt hoch, bzw. Aktor ist eingeschaltet** **3 = Aktor fährt runter, bzw. Aktor ist eingeschaltet**
	<int>	Value	**Aktueller Wert / Position des Aktors (0 bis 65535)**
	<int>	TargetValue	**Zielwert des Aktors (0 bis 65535) falls der Aktor in Bewegung ist.**
	<int>	Flags	**Maske der Zustandsbits (Details siehe selve.GW.device.getValues)**
	<int>	DayMode	**0 = Unbekannter Tageszustand** **1 = Aktor befindet sich im Nachtmodus** **2 = Aktor befindet sich in der Morgendämmerung** **3 = Aktor befinde sich im Tagmodus** **4 = Aktor befindet sich in der Abenddämmerung**
	<string>	Aktorname	**Aktueller Name des Aktors (UTF-8 formatiert).**
	<int>	Konfiguration	**Aktuell konfigurierter Typ des Aktors (Details siehe selve.GW.device.getInfo)**
Parameter: <methodResponse>	**---**		

10.2 selve.GW.event.sensor

Methode:	**selve.GW.event.sensor**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt, wenn die Funktion in den Gatewayeinstellungen aktiviert ist!** Mit Hilfe der Methode „selve.GW.event.sensor“ wird der Anwendung aktualisierte Sensordaten automatisch mitgeteilt. **HINWEIS Details zu den einzelnen Parametern sind in der Methodenklasse selve.GW.sensor zu entnehmen.**		
Parameter: <methodCall>	**<int>**	SensorID	**Nummer des Sensors (0 bis 7) zu dem die folgenden Daten gehören.**
	<int>	WindDigital	**Windwert (Details siehe Kap.5.1.3)**
	<int>	RainDigital	**Regenwert (Details siehe Kap.5.1.3)**
	<int>	TempDigital	**Temperturbereich (Siehe Kap.5.1.3)**
	<int>	LightDigital	**Helligkeit (Siehe Kap.5.1.3)**
	<int>	SensorStatus	**0 = Noch keine gültigen Sensordaten verfügbar** **1 = Sensordaten verfügbar** **2 = Sensordaten verfügbar, der Sensor meldet eine schwache Batterie.** **3 = Die Kommunikation zum Sensor ist abgebrochen – „Sensorverlust“.** **4 = Testmodus im Sensor aktiv** **5 = Servicemodus im Sensor aktiv**
	<int>	TempAnalog	**Analoger Temperaturwert (Siehe Kap.5.1.3)**
	<int>	WindAnalog	**Analoger Windwert (Siehe Kap.5.1.3)**
	<int>	Sun1Analog	**Analoger Sonnenwert (Siehe Kap.5.1.3)**
	<int>	DayLightAnalog	**Analoger Tageslichtwert (Siehe Kap.5.1.3)**
	<int>	Sun2Analog	**Analoger Wert des zweiten Sonnensensors (Siehe Kap.5.1.3)**
	<int>	Sun3Analog	**Analoger Wert des dritten Sonnensensors (Siehe Kap.5.1.3)**
Parameter: <methodResponse>	---		

10.3 selve.GW.event.sender

Methode:	**selve.GW.event.sender**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt, wenn die Funktion in den Gatewayeinstellungen aktiviert ist!** Mit Hilfe der Methode „selve.GW.event.sender“ wird der Anwendung Senderereignisse automatisch mitgeteilt. **HINWEIS Details zu den einzelnen Parametern sind in der Methodenklasse selve.GW.sender zu entnehmen.**		
Parameter: <methodCall>	**<int>**	SenderID	**Nummer des Senders (0 bis 62) zu dem die folgenden Daten gehören.**
	<int>	Ereignis	**Neues, vorliegendes Ereignis des Senders (Siehe Kap.7.1.4)**
	<string>	Name	**Aktueller Name des Senders (UTF-8 formatiert).**
Parameter: <methodResponse>	---		

10.4 selve.GW.event.log

Methode:	**selve.GW.event.log**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt, wenn die Funktion in den Gatewayeinstellungen aktiviert ist!** Mit Hilfe der Methode „selve.GW.event.log“ werden der Anwendung interne Gatewayereignisse mitgeteilt, die die Anwendung – falls vorhanden – in den internen Log-Speicher ablegen sollte. Die Meldungen unterscheiden sich zwischen folgenden Typen: **Information** Das Gateway sendet Meldungen als zusätzliche Informationen, die die Anwendung für eine Rückverfolgbarkeit speichern sollte. (z.B. Einlernprozesse, Werkszustände, etc.) **Warning** Das Gateway sendet Warnungen, die eine Anwendung für die Rückverfolgbarkeit speichern sollte. (Z.B. fehlerhafte Fahrkommandos, etc.) **Error** Das Gateway meldet Fehler, wenn z.B. Aufrufe nicht durchgeführt werden konnten.		
Parameter: <methodCall>	**<int>**	LogType	**0 = Information** **1 = Warning** **2 = Error**
	<string>	LogCode	**Intern generierter Code-Schlüssel des Logs.**
	<string>	LogStamp	**Aktueller, interner Zeitstempel des Logs.**
	<string>	LogValue	**Gesonderter, zusätzlicher Parameter für eine genauere Zuordnung im Gateway.**
	<string>	LogDescription	**Beschreibungstext des Logs**
Parameter: <methodResponse>	---		

10.5 selve.GW.event.dutyCycle

Methode:	**selve.GW.event.dutyCycle**		
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt, wenn die Funktion in den Gatewayeinstellungen aktiviert ist!** Mit Hilfe der Methode „selve.GW.event.dutyCycle“ wird der Anwendung die aktuell erlaubte Funkauslastung mitgeteilt.		
Parameter: <methodCall>	**<int>**	Mode	**0 = Senden wird nicht blockiert.** **1 = Senden wird blockiert.**
	<int>	Auslastung	**Auslastung des erlaubten Duty Cycles nach der Funkrichtlinie (0 bis 100 %)**
Parameter: <methodResponse>	---		

11 selve.GW.iveo – IVEO Funk

Die Methodenklasse **selve.GW.iveo** beinhaltet die Methoden, die es erlauben mit dem Gateway das unidirektionale SELVE Funksystem **iveo** bedienen zu können.

Die Methoden erlauben das Einlernen, bzw. Auslernen des Gateways in SELVE **iveo** Funkaktore, sowie die einfache Bedienung.

Das Gateway bietet 64 separate Funkkanäle, womit bis zu 64 **iveo** Aktore einzeln bedient werden können. Jeder Funkkanal wird mit einer eindeutigen IveoID (0 bis 63) im Gateway gekennzeichnet.

Die folgende Übersicht zeigt die zur Verfügung stehenden Methoden:

Methode	Kurzbeschreibung
`selve.GW.iveo.setRepeater`	Setzen der Konfiguration der im iveo System genutzten iveo Repeater.
`selve.GW.iveo.getRepeater`	Lesen der aktuellen Konfiguration der iveo Repeater Einstellung.
`selve.GW.iveo.setConfig`	Konfiguration einzelner iveo Funkkanäle.
`selve.GW.iveo.getConfig`	Lesen der aktuellen Konfiguration einzelner iveo Funkkanäle
`selve.GW.iveo.getIDs`	Lesen einer Maske, die zeigt welche IveoIDs aktiv genutzt werden.
`selve.GW.iveo.commandTeach`	Senden des Kommandos zum ein-, bzw. auslernen eines iveo Funkkanals.
`selve.GW.iveo.commandLearn`	Senden des Kommandos um die Aktore eines Funkkanals in Lernbereitschaft zu versetzen.
`selve.GW.iveo.commandManual`	Senden eines manuellen Bedienkommandos.
`selve.GW.iveo.commandAutomatic`	Senden eines automatischen Bedienkommandos für z.B. Schaltuhrfunktionen.
`selve.GW.iveo.commandResult`	Das Gateway meldet der Anwendung ein komplett durchgeführtes Bedienkommando.

Tabelle 15. *Übersicht der Methodenklasse „selve.GW.iveo“*

11.1 Grundlagen / Abläufe

Dieser Abschnitt beschreibt Abläufe und liefert detaillierte Informationen, die für die Bedienung der iveo Aktore hilfreich sind.

11.1.1 Ablauf einer Bedienung

Der folgende Ablauf zeigt das korrekte Handling eines Bedienkommandos. Es ermöglicht der Anwendung gleichzeitig eine zeitliche, optimierte Bedienung.

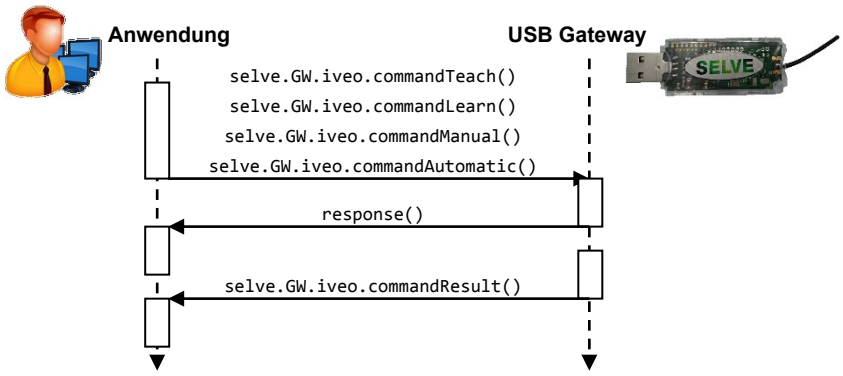

Bild 8. Grafische Darstellung des Ablaufs eines iveo Bedienkommandos.

11.1.2 Einlernen des Gateways

Das Einlernen eines Funkkanals in einen iveo Aktor erfolgt in folgenden Schritten:

- Der iveo Aktor wird mit Hilfe eines bereits eingelernten iveo Senders in die Lernbereischaft versetzt. Hierzu wird am iveo Sender für 3 Sekunden die PROG Taste betätigt, der Aktor signalisiert daraufhin die Lernbereitschaft mit einer kurzen Auf-, und Abfahrt. Die Lernbereitschaft ist nun für 1 Minute aktiv.
- Mit Hilfe der Methode **selve.GW.iveo.commandTeach** wird nun der Funkkanal des Gateways in den Aktor eingelernt.
- Mit jedem neu eingelernten Sender/Funkkanal muss der iveo Aktor erneut in den Lernmodus versetzt werden, so dass für das Einlernen mehrerer Funkkanäle der Ablauf wiederholt werden muss.

11.1.3 Auslernen des Gateways

Das Auslernen eines Funkkanals aus einen iveo Aktor erfolgt analog zum Einlernen in folgenden Schritten:

- Der iveo Aktor wird mit Hilfe eines **anderen** eingelernten iveo Senders in die Lernbereitschaft versetzt. Hierzu wird am iveo Sender für 3 Sekunden die PROG Taste betätigt, der Aktor signalisiert daraufhin die Lernbereitschaft mit einer kurzen Auf-, und Abfahrt. Die Lernbereitschaft ist nun für 1 Minute aktiv.
- Mit Hilfe der Methode **selve.GW.iveo.commandTeach** wird nun der aktuell eingelernte Funkkanal des Gateways aus dem Aktor ausgelernt. Mit dem Auslernen des Funkkanals wird der Lernmodus im **iveo** Aktor automatisch beendet.
- Für das Aus-, bzw. erneute Einlernen muss der entsprechende Aktor stets nach dem obigen Ablauf in den Einlernzustand versetzt werden.

11.1.4 Lernbereitschaft von Aktore starten

Ist ein Funkkanal in einem **iveo** Aktor eingelernt, so besteht die Möglichkeit diesen Aktor vom Gateway aus direkt in die Lernbereitschaft zu bringen.

Hierzu wird über den eingelernten Funkkanal die Methode **selve.GW.iveo.commeoLearn** aufgerufen. Die **iveo** Aktore signalisieren daraufhin die Lernbereitschaft mit einer kurzen Auf-, und Abfahrt und stehen für 1 Minute zum ein-, bzw. auslernen bereit.

HINWEIS Der Aktor, der die Lernbereitschaft startet kann sich selbst nicht auslernen!

11.2 selve.GW.iveo.setRepeater

Methode:	**selve.GW.iveo.setRepeater**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.setRepeater“ kann die Einstellung des Repeaterlevels des iveo Systems konfiguriert werden. Je nach eingestelltem Repeater-Level sorgt das Gateway selbstständig dafür, entsprechende Ruhezeiten zwischen einzelnen Kommandos einzuhalten, damit im **iveo** System installierte Repeater ihre Weiterleitung durchführen können. **HINWEIS Je höher der Repeater-Level eingestellt ist, desto länger dauert die Verarbeitung von iveo Funktelegrammen!**		
Parameter: <methodCall>	**<int>**	Repeater	**0 = Es ist kein Repeater im iveo System installiert.** **1 = Ein iveo Repeater für eine einfache Weiterleitung ist im iveo System installiert.** **2 = Mehrere iveo Repeater für zweifache Weiterleitungen sind im iveo System installiert.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.setRepeater**
	<int>	Return	**0 = Das Schreiben der Konfiguration ist fehlgeschlagen.** **1 = Das Schreiben der Konfiguration war erfolgreich.**

11.3 selve.GW.iveo.getRepeater

Methode:	**selve.GW.iveo.getRepeater**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.getRepeater“ kann die aktuelle Einstellung des Repeaterlevels gelesen werden.		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.getRepeater**
	<int>	Repeater	**0 = Es ist kein Repeater im iveo System installiert.** **1 = Ein iveo Repeater für eine einfache Weiterleitung ist im iveo System installiert.** **2 = Mehrere iveo Repeater für zweifache Weiterleitungen sind im iveo System installiert.**

11.4 selve.GW.iveo.setLabel

Methode:	**selve.GW.iveo.setLabel**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.setLabel“ kann der Name des Iveo Kanals im Gateway geändert werden. Dieser Name ist eine Bezeichnung, die nur im Gateway gespeichert wird.		
	HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63).**
	<string>	Name	**Neuer Name des iveo Funkkanals (Maximal 23 Byte)**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.setLabel**
	<int>	Return	**0 = Der Name konnte nicht übernommen werden.** **1 = Der Name konnte erfolgreich geändert werden.**

11.5 selve.GW.iveo.setConfig

Methode:	**selve.GW.iveo.setConfig**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.setConfig“ kann die Konfiguration einzelner **iveo** Funkkanäle durchgeführt werden.		
	Für eine Gleichheit zu den commeo Geräten werden intern Parameter wie Name und ein optional zu setzender Typ zur Verfügung gestellt. Diese dienen zur Beschreibung und sind nur intern im Gateway hinterlegt.		
	HINWEIS Die Namen liegen nach Spezifikation im UTF-8 Format vor. Die maximale Länge des Namens darf eine umgerechnete Bytelänge von 23 Byte nicht überschreiten!		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63).**
	<int>	Activity	**0 = Funkkanal wird deaktiviert und steht nicht mehr zur Verfügung.** **1 = Funkkanal wird aktiviert und steht den weiteren Kommandos zur Verfügung.**
	<int>	Typ	**0 = Kein Typ definiert** **1 = Rollladen** **2 = Jalousie** **3 = Markise** **4 = Schaltaktor** **5 = Dimmer** **6 = Nachtlicht Aktor** **7 = Dämmerlicht Aktor** **8 = Heizung** **9 = Kühlgerät** **10 = Schaltaktor (Tagbetrieb)** **11 = Gateway**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.setConfig**
	<int>	Return	**0 = Das Schreiben der Konfiguration ist fehlgeschlagen.** **1 = Das Schreiben der Konfiguration war erfolgreich.**

11.6 selve.GW.iveo.getConfig

Methode:	**selve.GW.iveo.getConfig**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.getConfig“ kann die aktuelle Konfiguration des ausgewählten Funkkanals gelesen werden. Details zu den einzelnen Parametern sind aus der Beschreibung der Methode selve.GW.iveo.setConfig zu entnehmen.		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63), der gelesen werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.getConfig**
	<int>	IveoID	**ID des iveo Funkkanals (0 bis 63).**
	<int>	Activity	**0 = Funkkanal ist deaktiviert.** **1 = Funkkanal ist aktiviert.**
	<string>	Name	**Name des Funkkanals (Maximal 23 Byte)**
	<int>	Typ	**Definierter Typ des Funkkanals (siehe oben)**

11.7 selve.GW.iveo.getIDs

Methode:	**selve.GW.iveo.getIDs**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.getIDs“ kann eine Maske gelesen werden, die zeigt welche IveoIDs im Gateway aktiviert und verwendet werden. Diese Methode ermöglicht einen optimierten Zugriff. Wird z.B. während der Initialisierungphase einer Anwendung alle Daten erfasst, so kann durch diese Maske das Lesen der notwendigen Daten beschleunigt werden. Die Maske der IveoIDs ist genau 8 Bytes lang. Jedes Bit repräsentiert eine IveoID. Ist das Bit auf 1 gesetzt, so handelt es sich bei dieser IveoID um einen genutzten Funkkanal. Eine 0 zeigt, dass diese IveoID nicht genutzt wird und bei z.B. einem kompletten Download nicht relevant ist. Byte 0.0 IveoID 0 ... Byte 0.7 IveoID 7 Byte 1.0 IveoID 8 ... Byte 7.7 IveoID 63		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.getIDs**
	<base64>	Maske	**Maske der genutzten IveoIDs**

11.8 selve.GW.iveo.factory

Methode:	**selve.GW.iveo.factory**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.factory“ kann ein iveo Funkkanal komplett gelöscht und auf Werkszustand gesetzt werden. **HINWEIS Bevor der iveo Kanalzurückgesetzt wird, sollte der Funkkanal aus allen Aktoren ausgelernt werden, damit eine spätere, fehlerhafte Bedienung ausgeschlossen werden kann.**		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63), der gelöscht werden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.factory**
	<int>	Return	**0 = Ein Fehler beim Löschen ist aufgetreten.** **1 = Das Löschen konnte erfolgreich durchgeführt werden.**

11.9 selve.GW.iveo.commandTeach

Methode:	**selve.GW.iveo.commandTeach**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.commandTeach“ kann ein iveo Funkkanal des Gateways in einen iveo Aktor ein-, bzw. ausgelernt werden. Der Ein-, bzw. Auslernprozess ist im Kapitel 11.1 detailliert beschrieben.		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63), der ein Lernkommando senden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.commandTeach**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.** **1 = Das Kommando wird gesendet, die vollständige Durchführung des Kommandos wird über das Result mitgeteilt.**

11.10 selve.GW.iveo.commandLearn

Methode:	**selve.GW.iveo.commandLearn**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.commandLearn“ kann ein iveo Funkkanal des Gateways einen bereits eingelernten Aktor mitteilen, dass er die Lernbereitschaft starten soll. Die Lernbereitschaft wird vom iveo Aktor entsprechend angezeigt, i.d.R. durch ein kurze Auf-, und Abfahrt.		
Parameter: <methodCall>	**<int>**	IveoID	**ID des iveo Funkkanals (0 bis 63), das Kommando zum starten der Lernbereitschaft senden soll.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.commandLearn**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.** **1 = Das Kommando wird gesendet, die vollständige Durchführung des Kommandos wird über das Result mitgeteilt.**

11.11 selve.GW.iveo.commandManual

Methode:	**selve.GW.iveo.commandManual**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.commandManual“ kann eine Gruppe von IveoIDs mit einem manuellen Kommando angesprochen werden. Die einzelnen Gruppenmitglieder werden in einer Maske von genau 8 Bytes dem Gateway mitgeteilt. Jedes Bit repräsentiert eine IveoID. Ist das Bit auf 1 gesetzt, so wird das Kommando über den entsprechenden Funkkanal gesendet, mit einer 0 ist die IveoID für das Kommando nicht vorgesehen. Byte 0.0 IveoID 0 ... Byte 0.7 IveoID 7 Byte 1.0 IveoID 8 ... Byte 7.7 IveoID 63 **HINWEIS Die einzelnen Funkkanäle werden sequential abgearbeitet. Je nach Anzahl der Gruppenmitglieder und des eingestellten Repeater-Levels wird die Abarbeitung unterschiedliche Durchführungszeiten haben.** **Ein neues Kommando unterbricht stets die Abarbeitung des laufenden Kommandos. Die Anwendung muss dies berücksichten und sollte i.d.R. auf das Ende (selve.GW.iveo.commandResult) warten.**		
Parameter: <methodCall>	**<base64>**	Maske	**Anwendungsspezifische Maske der IveoIDs, die das Kommando ausführen sollen.**
	<int>	Kommando	**0 = Stoppen der Aktore** **1 = Auffahrt der Aktore** **2 = Abfahrt der Aktore** **3 = Anfahrt der im Aktor eingestellten Zwischenposition 1** **4 = Anfahrt der im Aktor eingestellten Zwischenposition 2**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.iveo.commandManual**
	<int>	Return	**0 = Kommando konnte nicht ausgeführt werden.** **1 = Das Kommando wird gesendet, die vollständige Durchführung des Kommandos wird über das Result mitgeteilt.**

11.12 selve.GW.iveo.commandAutomatic

Methode:	**selve.GW.iveo.commandAutomatic**
Beschreibung:	Mit Hilfe der Methode „selve.GW.iveo.commandAutomatic“ kann eine Gruppe von IveoIDs mit einem automatischen Kommando angesprochen werden. Diese werden nur ausgeführt, wenn in den Aktoren der Automatikbetrieb aktiviert ist. Diese Methode dient z.B. für automatische Fahrten, die durch Schaltuhren ausgeführt werden. Die einzelnen Gruppenmitglieder werden in einer Maske von genau 8 Bytes dem Gateway mitgeteilt. Jedes Bit repräsentiert eine IveoID. Ist das Bit auf 1 gesetzt, so wird das Kommando über den entsprechenden Funkkanal gesendet, mit einer 0 ist die IveoID für das Kommando nicht vorgesehen. Byte 0.0 IveoID 0 ... Byte 0.7 IveoID 7 Byte 1.0 IveoID 8 ... Byte 7.7 IveoID 63 **HINWEIS Die einzelnen Funkkanäle werden sequential abgearbeitet. Je nach Anzahl der Gruppenmitglieder und des eingestellten Repeater-Levels wird die Abarbeitung unterschiedliche Durchführungszeiten haben.** **Ein neues Kommando unterbricht stets die Abarbeitung des laufenden Kommandos. Die Anwendung muss dies berücksichten und sollte i.d.R. auf das Ende (selve.GW.iveo.commandResult) warten.**
Parameter: <methodCall>	**<base64>** Maske — **Anwendungsspezifische Maske der IveoIDs, die das Kommando ausführen sollen.** **<int>** Kommando — **0 = Stoppen der Aktore** **1 = Auffahrt der Aktore** **2 = Abfahrt der Aktore** **3 = Anfahrt der im Aktor eingestellten Zwischenposition 1** **4 = Anfahrt der im Aktor eingestellten Zwischenposition 2**
Parameter: <methodResponse>	**<string>** Methodenname — **selve.GW.iveo.commandAutomatic** **<int>** Return — **0 = Kommando konnte nicht ausgeführt werden.** **1 = Das Kommando wird gesendet, die vollständige Durchführung des Kommandos wird über das Result mitgeteilt.**

11.13 selve.GW.iveo.commandResult

Methode:	**selve.GW.iveo.commandResult**
Beschreibung:	**HINWEIS Diese Methode wird als Ereignis automatisch vom Gateway erzeugt!** Mit Hilfe der Methode „selve.GW.iveo.commandResult“ wird der Anwendung mitgeteilt, dass das iveo Kommando komplett abgearbeitet wurde. Für eine schnelle Zuordnung wird in der Antwort das ausgeführte Kommando mitgeliefert. Weiterhin zeigt die Maske welche IveoIDs bedient wurden. Byte 0.0 IveoID 0 ... Byte 0.7 IveoID 7 Byte 1.0 IveoID 8 ... Byte 7.7 IveoID 63
Parameter: <methodCall>	**<int>** Kommando **0 = Stoppen der Aktore** **1 = Auffahrt der Aktore** **2 = Abfahrt der Aktore** **3 = Anfahrt der im Aktor eingestellten Zwischenposition 1** **4 = Anfahrt der im Aktor eingestellten Zwischenposition 2** **254 = Learn Telegramm wurde gesendet** **255 = Teach Telegramm wurde gesendet** **<base64>** Maske **Maske der IveoIDs, die das Kommando erhalten haben.** **<int>** State **0 = IDLE Status, es wird kein iveo Telegramm gesendet, bzw. das Senden des Telegramms ist beendet.** **1 = SEND Status, das iveo Telegramm wird gerade gesendet.**
Parameter: <methodResponse>	---

12 selve.GW.firmware – Gateway Bootloader

Die Methodenklasse **selve.GW.firmware** beinhaltet die Methoden, die es erlauben die Firmware des USB Gateways zu aktualisieren.

Für eine erfolgreiche Aktualisierung der Firmware muss der folgende Ablauf eingehalten werden. Die Daten aus der Firmwaredatei müssen vollständig und fehlerfrei übertragen werden!

Der Bootloaderbereich des Gateways ist geschützt und sorgt gleichzeitig dafür, dass bei einer fehlerhaften oder unvollständigen Firmwareaktualisierung das Gateway weiterhin erreichbar bleibt. Nach jedem Neustart des Gateways wird der Bootbereich für 2 Sekunden durchlaufen, bevor die eigentliche Firmware des Gateways gestartet wird. In dieser Zeit besteht die Möglichkeit das Firmwareupdate zu starten.

Die folgende Übersicht zeigt die Methoden, die für das Updaten der Firmware notwendig sind. Zu den in diesem Kapitel beschriebenen Methoden gehören Methoden aus der Klasse selve.GW.service.

Befindet sich das Gateway – angezeigt durch das Aufleuchten der roten LED – wegen eines Updates oder einer unvollständigen Firmwareaktualisierung im Bootloader, so stehen der Anwendung weiterhin die unten beschriebenen Methoden zur Verfügung.

Die folgende Übersicht zeigt die zur Verfügung stehenden Methoden des Bootloaders:

Methode	Kurzbeschreibung
`selve.GW.service.ping`	Ping zum SELVE USB-RF Gateway. (Details sind aus der entsprechenden Methodenbeschreibung zu entnehmen!)
`selve.GW.service.getState`	Lesen des aktuellen Status des Gateways. (Details sind aus der entsprechenden Methodenbeschreibung zu entnehmen!)
`selve.GW.service.reset`	Neustart des SELVE USB-RF Gateways. (Details sind aus der entsprechenden Methodenbeschreibung zu entnehmen!)
`selve.GW.firmware.start`	Befindet sich das Gateway im Bootloaderbereich, kann durch diese Methode das Firmwareupdate gestartet werden.
`selve.GW.firmware.data`	Während eines Firmwareupdates werden mit dieser Methode die Firmwaredaten übertragen.
`selve.GW.firmware.end`	Beendet ein vollständiges Firmwareupdate.

Tabelle 16. *Übersicht der Methodenklasse „selve.GW.firmware“*

12.1 Ablauf eines Firmwareupdates

Dieser Abschnitt beschreibt den kompletten Ablauf eines Firmwareupdates. Dieser Ablauf muss durch die Anwendung durchgeführt werden, um ein korrektes, vollständiges Update zu gewährleisten.

Im Falle eines Fehlers kann jedoch das Firmwareupdate jederzeit neu gestartet werden.

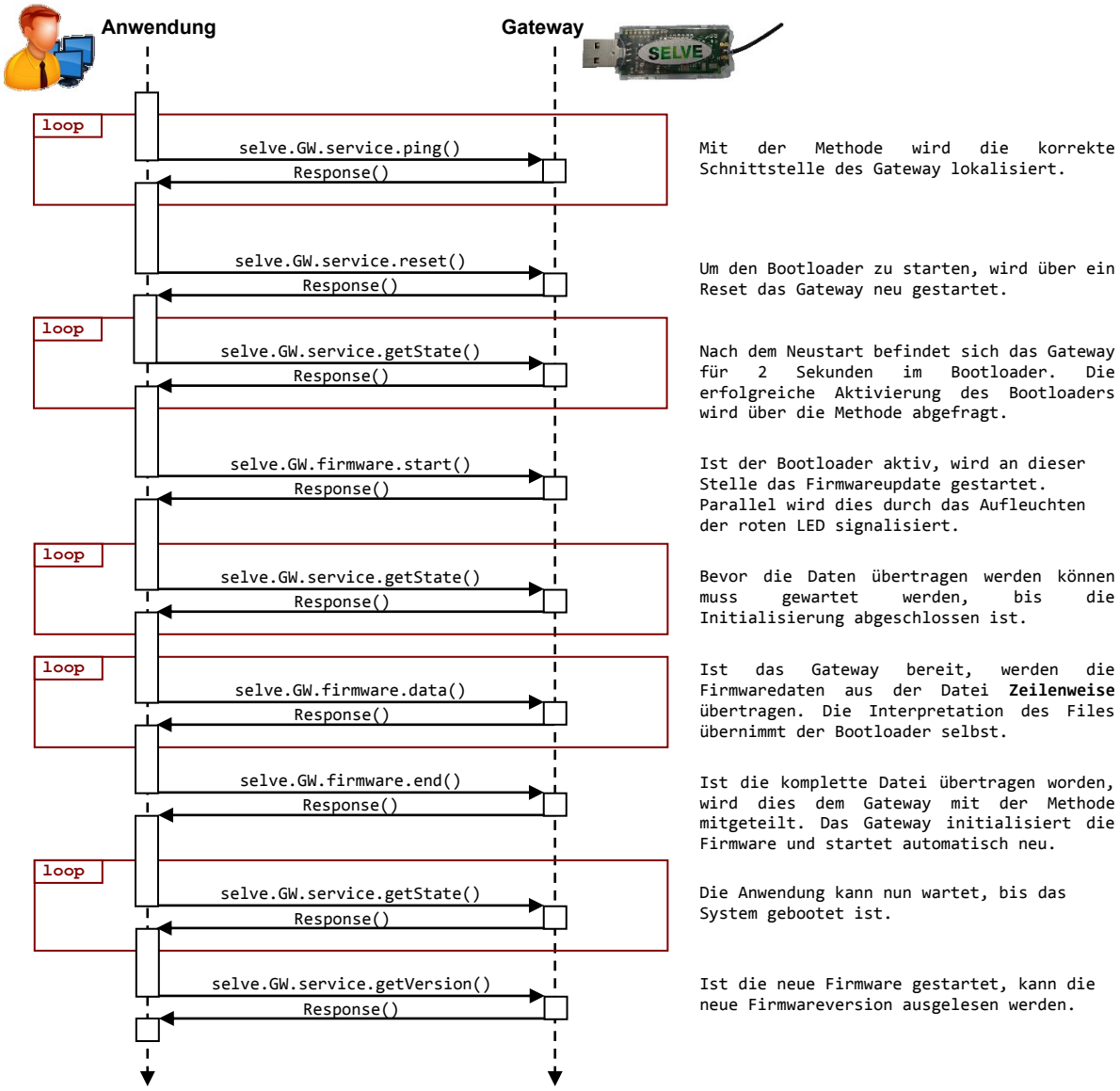

Bild 9. Grafische Darstellung des Ablaufs einer Firmwareaktualisierung.

12.2 selve.GW.firmware.start

Methode:	**selve.GW.firmware.start**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.firmware.start“ wird ein neues Firmwareupdate gestartet. **HINWEIS Diese Methode kann nur aufgerufen werden, wenn sich das Gateway im Bootloader befindet. Die eigentliche Firmware unterstützt diese Methode nicht und quittiert dies mit dem entsprechenden Fehler.**		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.firmware.start**
	<int>	Return	**0 = Der Bootloader meldet, dass das Firmwareupdate nicht gestartet werden konnte.** **1 = Der Bootloader initialisiert ein neues Firmwareupdate.**

12.3 selve.GW.firmware.end

Methode:	**selve.GW.firmware.end**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.firmware.end“ wird das Ende eines Firmwareupdates signalisiert. **Diese Methode darf erst aufgerufen werden, wenn die Firmwaredatei komplett übertragen wurde. Ansonsten bleibt das Gateway im Bootloader und ist nicht für den normalen Betrieb bereit.** **HINWEIS Diese Methode kann nur aufgerufen werden, wenn sich das Gateway im Bootloader befindet. Die eigentliche Firmware unterstützt diese Methode nicht und quittiert dies mit dem entsprechenden Fehler.**		
Parameter: <methodCall>	---		
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.firmware.end**
	<int>	Return	**0 = Der Bootloader meldet, dass ein Neustart nicht durchgeführt werden konnte.** **1 = Der Bootloader initialisiert die Firmware und startet das Gateway neu.**

12.4 selve.GW.firmware.data

Methode:	**selve.GW.firmware.data**		
Beschreibung:	Mit Hilfe der Methode „selve.GW.firmware.data“ werden die Daten der Firmwaredatei übertragen. **Diese Methode darf erst aufgerufen werden,der Bootloader den Start der Firmwareaktualisierung initialisiert hat!** **HINWEIS Pro Aufruf wird immer genau eine Zeile der Firmwaredatei, ohne den Zeilenumbruch (CR LF) übertragen.** **HINWEIS Die Reihenfolge der Firmwaredatei ist zwingend einzuhalten!** **HINWEIS Diese Methode kann nur aufgerufen werden, wenn sich das Gateway im Bootloader befindet. Die eigentliche Firmware unterstützt diese Methode nicht und quittiert dies mit dem entsprechenden Fehler.**		
Parameter: <methodCall>	**<string>**	FirmwareZeile	**Einzelne Firmwarezeilen werden sequenziell während des Firmwareupdates übertragen. Pro Paket wird immer genau eine Zeile der Datei übertragen.**
Parameter: <methodResponse>	**<string>**	Methodenname	**selve.GW.firmware.data**
	<int>	Return	**0 = Die Firmwaredaten konnten nicht entpackt werden.** **1 = Die Firmwaredaten sind entpackt und konnten erfolgreich geschrieben werden.**

Beispiel:

```
<methodCall>
        <methodName>selve.GW.firmware.data</methodName>
        <array>
                <string>B7B7.... DF</string>
        </arry>
</methodCall>
```


```
<?xml version="1.0"?>
<methodResponse>
        <array>
                <string>selve.GW.firmware.data</string>
                <int>1</int>
        </array>
</methodResponse>
```

Anhang A: Fehlermeldungen

Die folgenden Fehlercodes sind für das Gateway definiert und erleichtern der Anwendung die Fehleranalyse.

Code	Text	Beschreibung
1	Unknown Error!	**Ein allgemeiner, nicht näher beschriebene Fehler ist aufgetreten und die gewünschte Aktion wurde nicht ausgeführt.**
2	Method not supported!	**Die aufgerufene Methode wird nicht unterstützt.** (Ein falscher Methodenname wurde gewählt, z.B. „selve.GW.test“).
3	Method not reachable!	**Diese Fehlermeldung wird generiert, wenn zwar die Funktion grundsätzlich implementiert ist, aber durch den aktuellen Status nicht erreichbar ist.** Z.B. In der eigentlichen Firmware können die Methoden des Bootloaders selve.GW.firmware nicht aufgerufen werden.
4	Parameter count!	**Die Anzahl der übergebenen Parameter stimmt nicht mit der aus der Definition überein. Die Methode wurde daher nicht ausgeführt.** Z.B. Die Methode „selve.GW.param.setForward“ erwartet einen Parameter. Wird hier kein Parameter gesendet, so wird dies mit diesem Fehlercode quittiert.
5	Parameter order!	**Die Reihenfolge der Parameter ist vertauscht.** Z.B. Bei der Auswertung ist anhand der Reihenfolge <string>, <int>, <base64> erkannt worden, dass die korrekte Reihenfolge nicht eingehalten wurde.
6	Execution failed!	**Es ist ein Fehler bei der Ausführung der Methode entstanden.** Z.B. Wird gerade ein Einlernprozess eines Aktors durchgeführt, so kann in dieser Zeit kein weiterer Einlernprozess gestartet werden.
7	Parameter out of range!	**Ein übergebener Parameter ist außerhalb des definierten Bereiches.** Z.B. Ein Parameter, der für einen Wertebereich von 0 bis 63 definiert wurde, ist auf 64 gesetzt worden.
8	Syntax error!	**Ein Syntax Fehler beim Aufbau der Methode liegt vor.** Z.B. Ist ein Parameter <int>, <string>, <base64> außerhalb des Arrays <array> übergeben, wird dies als Syntax Fehler quittiert.
9	Method length too large!	**Aufgrund einer zu langen Formatierung odr falschen Anweisung wird der im Gateway vorgesehende Empfangspuffer überschritten.**
10	ID is not used!	**Die übergebene ID wird nicht verwendet, so dass die gewünschte Aktion nicht ausgeführt werden kann.** Z.B. Wenn eine AktorID eingespeichert werden soll, die intern gar nicht verwendet wird.
11	ID is already exists!	**Die aufgerufene ID wird bereits genutzt und kann nicht beschrieben werden.** Z.B. Über die Methode selve.GW.device.writeManual wird eine AktorID beschrieben, die bereits genutzt wird.
12	Address is already used!	**Die übergebene Adresse wird bereits genutzt, die Aktion kann nicht durchgeführt werden.** Z.B. Über die Methode selve.GW.device.writeManual wird eine Funkadresse geschrieben, die bereits unter einer anderen AktorID genutzt wird.
13	No member available!	**Das Bedienkommando enthält keine aktiv genutzte AktorID, so dass das Kommando nicht ausgeführt wird.** Z.B. In einem Gruppenkommando sind nur Gruppenmitglieder gesetzt, die nicht als aktive AktorIDs eingetragen sind. Daher bleibt nach der Filterung kein Aktor mehr übrig, der noch bedient werden müsste.

14	`Duty Cycle is Reached!`	**Der erlaubte Grenzwert des Duty Cycles ist erreicht und das Telegram wurde daher nicht gesendet.**
40	`Bootl: Method not supported!`	**Befindet sich das Gateway im Bootloader, so sind nur die im Kapitel beschrieben Methoden verfügbar. Als einheitliche Meldung wird dies mit diesem Fehlercode quittiert.** Z.B. Das Gateway befindet sich im Bootloader und die Funktion selve.GW.command.device wird aufgerufen.
41	`Bootl: Wrong file!`	**Anhand der Datenanalyse wurde erkannt, dass es sich um fehlerhafte, oder falsche Firmwaredaten handelt.**
42	`Bootl: Checksum error!`	**Beim entpacken eines Datenpakets während eines Firmwareupdates hat der Bootloader einen CRC Fehler erkannt, so dass die Daten nicht geschrieben wurden.**
43	`Bootl: Syntax error!`	**Ein allgemeiner Syntaxfehler wurde im Bereich des Bootloader erkannt.**

Tabelle 17. *Übersicht der Fehlermeldungen*

Anhang B: Log-Übersicht

Im Folgenden werden die generierten Logging Codes näher beschrieben. Diese sollten, wenn die Anwendung es ermöglicht, gespeichert und zugänglich gemacht werden, um eine besser Analyse und Rückverfolgbarkeit zu gewährleisten. Bietet die Anwendung Uhrzeit und Datum, so kann zusammen mit den Meldungen der Verlauf des Gateways näher gedeutet werden.

Die folgende Übersicht beschreibt die verschieden Logging Codes des Gateways.

Code	Typ	Parameter	Beschreibung
0100	Information	---	**STATE: STARTUP - Initialization runs!** Wird das Gateway gestartet, so wird der Beginn der Startup-Phase als Log mitgeteilt.
0101	Information	---	**STATE: READY – Normal operation!** Nach der Startup-Phase ist das Gateway betriebsbereit, was als Log mitgeteilt wird.
0102	Information	---	**RESET: Restart is executed!** Das Gateway wird durch einen Softwarereset neugestartet.
0103	Information	**Version**	**FIRMWARE: New version (%1)!** Eine neue Firmwareversion ist erkannt worden.
0104	Information	---	**FACTORY: Gateway was resetted!** Das Gateway wurde auf Werkszustand zurückgesetzt.
0200	Warning	---	**DUTY CYCLE: Is Reached!** Die nach Richtlinie erlaubte Funkbelastung vom Gateway ist erreicht.
0201	Information	---	**DUTY CYCLE: OK!** Die Funkbelastung vom Gateway nimmt ab und ist wieder frei.
1000	Information	---	**DEVICE: Scan was started!** Ein neuer Suchlauf wurde gestartet.
1001	Information	**Aktor-adresse**	**DEVICE: Saving was successfully (Address: %1)!** Es wurde erfolgreich ein neuer Aktor im Gateway eingelernt (siehe Parameter).
1002	Error	**Aktor-adresse**	**DEVICE: Saving was failed (Address: %1)!** Das Einlernen des Aktors (siehe Parameter) ist fehlgeschlagen und konnte nicht erfolgreich durchgeführt werden.
1003	Warning	**Aktor-adresse**	**DEVICE: Device was deleted (Address: %1)!** Der Aktor wurde aus dem Gateway gelöscht, bzw. entfernt (siehe Parameter).
1004	Warning	**Aktor-adresse**	**DEVICE: Connection deleted (Address: %1)!** Das Gateway wurde aus dem Aktor gelöscht (Z.B. durch ein Reset des Aktors, etc.)
1005	Error	**Aktor-adresse**	**DEVICE: Connection lost (Address: %1)!** Es besteht keine Funkverbindung zum Aktor (siehe Parameter) mehr.
1006	Information	---	**DEVICE: Maximum device size is reached!** Während eines Suchlaufs wurde die maximale Anzahl an Aktore erreicht, so dass keine weiteren Aktore im Gateway mehr Platz finden.
1007	Warning	**Aktor-adresse**	**DEVICE: State is set to old (Address: %1)!** Der Löschbefehl konnte den Aktor nicht erreichen, so dass der Aktor im Gateway auf „OLD“ gesetzt wurde. (siehe Parameter).

1100	Information	**Sensor-adresse**	**SENSOR: New sensor saved (Address: %1)!** Ein neuer Sensor (siehe Parameter) wurde im Gateway eingelernt.
1101	Warning	**Sensor-adresse**	**SENSOR: Sensor was deleted (Address: %1)!** Die SensorID wurde aus dem Gateway gelöscht.
1102	Error	**Sensor-adresse**	**SENSOR: Connection lost (Address: %1)!** Es besteht keine Funkverbindung zum Sensor (siehe Parameter) mehr.
1200	Information	**Sender-adresse**	**SENDER: New sender saved (Address: %1)!** Ein neuer Sender (siehe Parameter) wurde im Gateway eingelernt.
1201	Warning	**Sender-adresse**	**SENDER: Sender was deleted (Address: %1)!** Die SenderID wurde aus dem Gateway ausgelernt, bzw. gelöscht.
1300	Warning	**Kommando**	**COMMEO: Radio line is busy (Command: %1)!** Das commeo Kommando wurde nicht ausgeführt, da bereits ein anderes Kommando durchgeführt wird.
1301	Warning	**Kommando**	**COMMEO: Command overwritten (Command: %1)!** Ein laufendes commeo Kommando (siehe Parameter) wurde abgebrochen und durch ein neues überschrieben.
1400	Warning	**Kommando**	**IVEO: Radio line is busy (Command: %1)!** Das iveo Kommando wurde nicht ausgeführt, da bereits ein anderes Kommando durchgeführt wird.
1401	Warning	**Kommando**	**IVEO: Command overwritten (Command: %1)!** Ein laufendes iveo Kommando (siehe Parameter) wurde abgebrochen und durch ein neues überschrieben.

Tabelle 18. *Übersicht der Logs*

SELVE GMBH & Co. KG
Werdohler Landstraße 286
D-58513 Lüdenscheid
Tel.: +49 2351 925-0
Fax: +49 2351 925-111
Internet: www.selve.de
E-Mail: info@selve.de