izinto – Partner für IoT

izi-io

Konfiguration und Payload Decoder

Dokumentversion:	1.1
Beschriebene Produktversion:	Firmware/Anwendung grid6
Dokumentdatum:	2023-03-20
Status:	Release
Freigabestatus:	**public**

Dokumenthistorie:

Version	Datum	Autor	Status	Beschreibung
0.1	2021-12-07	SK	in Bearbeitung	erstellt aus Vorgängerdokument
0.2	2022-01-15	SW, SK	in Bearbeitung	Beschreibung izi-io Parameter ergänzt
1.0	2023-03-17	SW, SK	in Bearbeitung	umstrukturiert; neue Parameter bis grid6
1.1	2023-03-20	SW, SK	Release	Korrekturen/Ergänzungen

Hinweise

Die Informationen in diesem Dokument wurden mit größter Sorgfalt erarbeitet. Dennoch können Fehler nicht vollständig ausgeschlossen werden. izinto übernimmt keine Haftung für eventuell verbliebene Fehler und deren Folgen.

Änderungen vorbehalten.

Alle Warennamen werden ohne Gewährleistung der freien Verwendbarkeit genutzt und sind möglicherweise eingetragene Warenzeichen.

Für Fragen und Kommentare stehen wir gerne zur Verfügung.

Inhalt

1 Einführung

Die Geräte der **izi-io**-Familie sind, abgesehen von der Nutzung individueller Firmwarelösungen, auch mit den verschiedenen angebotenen Standard-Firmwareversionen bereits in hohem Maße konfigurierbar und können an verschiedenste Bedingungen angepasst werden. Dazu kann ein integriertes Konfigurationsmanagement eingesetzt werden, in dessen Zentrum eine Konfigurationsdatei im JSON-Format steht. Diese Datei kann einerseits von der izi-io-Hardware eingelesen werden, die daraus ihre Parameter bezieht, welche Konfiguration der Ein- und Ausgänge, das Verhalten der Firmware und schließlich die LoRaWAN-Parameter und das verwendete Telegramm-Layout definieren. Andererseits wird die Datei von einem izinto-eigenen Tool namens *izi-confio* verarbeitet, das daraus eine Dokumentation als PDF sowie einen Payload Decoder generiert.

Der Payload Decoder ist eine JavaScript-Funktion, die im The Things Stack V3 für eine Application bzw. individuell für ein Device hinterlegt wird. Für jedes empfangene LoRaWAN-Telegramm wird diese Funktion aufgerufen. Sie liest das Telegramm ein und erzeugt einen Nutzdatensatz daraus, der dann über die Infrastruktur weitergegeben wird. Alternativ wird dieses Programm auch als Payload Formatter bezeichnet. In diesem Dokument werden beide Bezeichnungen synonym verwendet.

Dieses Dokument beschreibt die kundenspezifische Konfiguration von Geräten der **izi-io**-Familie über eine microSD-Karte, die hierzu erforderliche JSON-Konfigurationsdatei und den zugehörigen Payload Decoder. Außerdem wird die Nutzung des Tools izi-confio beschrieben.

Erläuterung der in diesem Dokument verwendeten Schrifttypen:

- *Kursive Schrift* dient der Hervorhebung einzelner Begriffe.
- **Fette Schrift** oder Unterstreichung wird zur Hervorhebung wichtiger Textstellen verwendet, etwa für Warnhinweise.
- `Schreibmaschinenschrift` markiert Texte, die auf dem Rechner verwendet werden, etwa Kommandonamen, Dateinamen etc. sowie Dateiinhalte.
- **`Fette Schreibmaschinenschrift`** markiert Schlüsselwörter in der JSON-Datei.
- *`Kursive Schreibmaschinenschrift`* steht für Platzhalter beispielsweise in einem Dateiinhalt, etwa einer JSON-Struktur, oder einem Kommando.

2 Durchführung Parameterupdate über die SD-Karte

Voraussetzungen:

- Verwendbar sind microSD-Karten, auch SDHC.
- Die SD-Karte muss FAT32-formatiert sein. ExFAT und SDXC werden nicht unterstützt.
- Die Konfigurationsdatei muss wie in Abschnitt 5 beschrieben aufgebaut sein.
- Die Konfigurationsdatei muss den Namen `config.json` haben und im Wurzelverzeichnis (oberste Ebene) der SD-Karte abgelegt sein.

Zur Konfiguration der izi-io sind folgende Schritte notwendig:

1. Das Gerät mit der Stromversorgung verbinden (noch keine SD-Karte eingesteckt).
2. Die SD-Karte wird mit den Kontakten in Richtung des schwarzen Streifens auf der Frontplatte der izi-io in den Kartenschacht eingeführt, bis sie einrastet. Zum Einrasten der Karte (hörbarer Klick) muss die Karte tief eingedrückt werden; das gelingt mit dem Fingernagel oder am besten mit einem Hilfsmittel wie einer Plastikkarte.
3. Die LED leuchtet gelb mit Unterbrechungen, solange die Datei auf der Karte geöffnet ist. **Die Karte darf in dieser Zeit nicht entfernt werden**.
4. Nach erfolgreichem Abschluss der Operation leuchtet die LED grün mit kurzen Unterbrechungen. Die izi-io wendet die neuen Parameter sofort an.
5. Die Karte wird entnommen, indem sie tiefer in den Kartenschacht gedrückt wird, bis sie wieder ausrastet und herausgezogen werden kann.

Falls die LED nach Abschluss des Vorgangs nicht grün leuchtet, ist ein Fehler aufgetreten:

- LED leuchtet magentafarben (violett) mit kurzen Unterbrechungen: für die izi-io mit dieser Seriennummer sind keine Parameterdaten auf der Karte vorhanden.
- LED leuchtet rot mit kurzen Unterbrechungen:
  - die Daten konnten aufgrund eines internen Fehlers nicht gelesen werden oder
  - die JSON-Datei enthält Syntaxfehler oder
  - die JSON-Datei enthält keine gültigen Parameter.
- LED blinkt rot: die SD-Karte konnte nicht initialisiert werden. Die Karte ist mit dem falschen Dateisystem formatiert oder defekt.

Wenn die LED rot leuchtet oder blinkt, ist es ratsam, die SD-Karte kurz zu entnehmen und wieder einzustecken. Der Vorgang startet dann erneut. Sollte auch nach mehreren Versuchen die Fehlermeldung bestehen bleiben, liegt eine fehlerhafte SD-Karte oder eine fehlerhafte Datei vor und die SD-Karte sollte mit einem Kartenlesegerät am Rechner überprüft werden.

Übersicht über die Farben und Blinkmuster der LED beim Einlesen der Konfiguration:

Farbe + Erscheinungsbild	**Betriebszustand**
gelb mit Unterbrechungen	SD-Karte wird bearbeitet
grün mit Unterbrechungen	SD-Kartenoperation abgeschlossen
magenta mit Unterbrechungen	keine Parameter auf SD-Karte gefunden
rot mit Unterbrechungen	Fehler in JSON-Konfiguration
rot blinkend	Fehler der SD-Karte

3 izi-confio

Das Konfigurationstool **izi-confio** ist ein Tool, das aus einer in JSON formulierten Konfiguration, die gleichermaßen direkt von der izi-io-Firmware eingelesen werden kann, sowohl eine Dokumentation als auch den dazugehörigen Payload Formatter (auch synonym als Payload Decoder bezeichnet) erzeugen kann.

Der Payload Formatter ist eine JavaScript-Funktion, die im Kontext der LoRaWAN-Infrastruktur The Thing Stack für eine Application bzw. individuell (nur V3) für ein Device eingetragen wird. Für jedes empfangene LoRaWAN-Telegramm wird die Funktion aufgerufen. Sie analysiert das Telegramm und erzeugt einen Nutzdatensatz daraus, der dann über die Infrastruktur weitergegeben wird. Für andere LoRaWAN-Server ist gegebenenfalls eine Portierung des JavaScript-Programms erforderlich – entweder, falls hier ebenfalls mit JavaScript gearbeitet wird, durch Umschreiben oder Verwendung eines Wrappers, oder durch Portierung in eine andere Programmiersprache. Hier kann izinto bei Bedarf unterstützen.

Die Dokumentation ist ein PDF-Dokument, das für ein einzelnes Device eine vollständige Beschreibung der Konfiguration darstellt. Die LoRaWAN-Schlüssel können dabei wahlweise mit dokumentiert oder ausgelassen werden, so dass das Dokument ggf. an Dritte weitergegeben werden kann, ohne diese sensiblen Daten offenzulegen.

izi-confio steht als Website zur Verfügung, auf der die JSON-Konfigurationsdatei hochgeladen wird. Anschließend kann in einem Formular die gewünschte Parametrierung vorgenommen und das Tool gestartet werden. Eine Ergebnisseite erlaubt schließlich den Download der generierten Artefakte.

3.1 Aufruf

Unter der Adresse https://confio.izi-cloud.de/upload kann das Upload-Formular aufgerufen werden. Dazu ist einmalig ein Website-Login erforderlich, für das die Zugangsdaten (Username und Password) von izinto bereitgestellt werden.

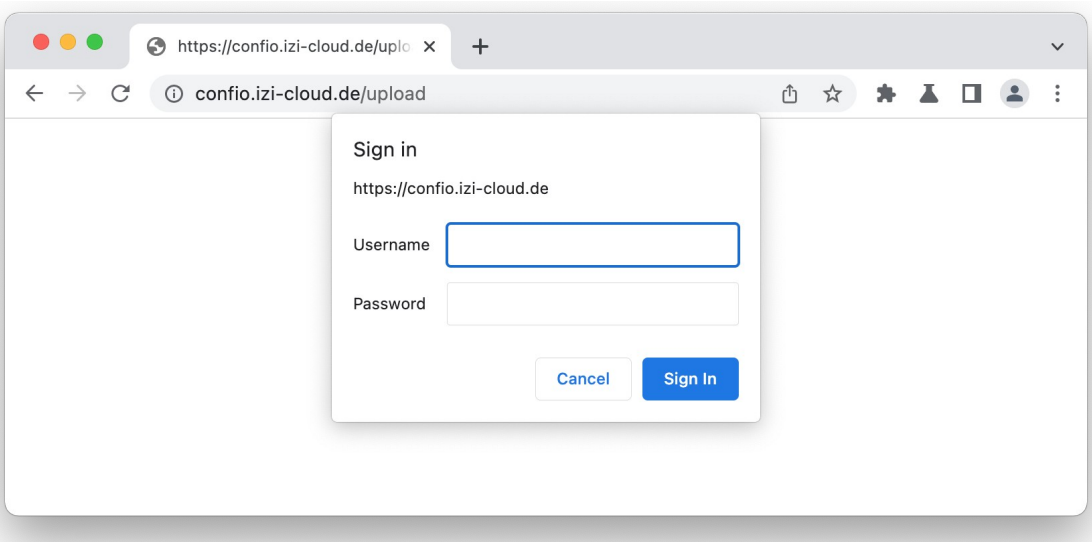

Abb. 1: Anmeldedialog

Nach erfolgreicher Anmeldung wird das Login in den Browserdaten (Cookies) gespeichert, so dass dieser Schritt zukünftig nicht mehr notwendig ist.

Anschließend wird das Upload-Formular gezeigt.

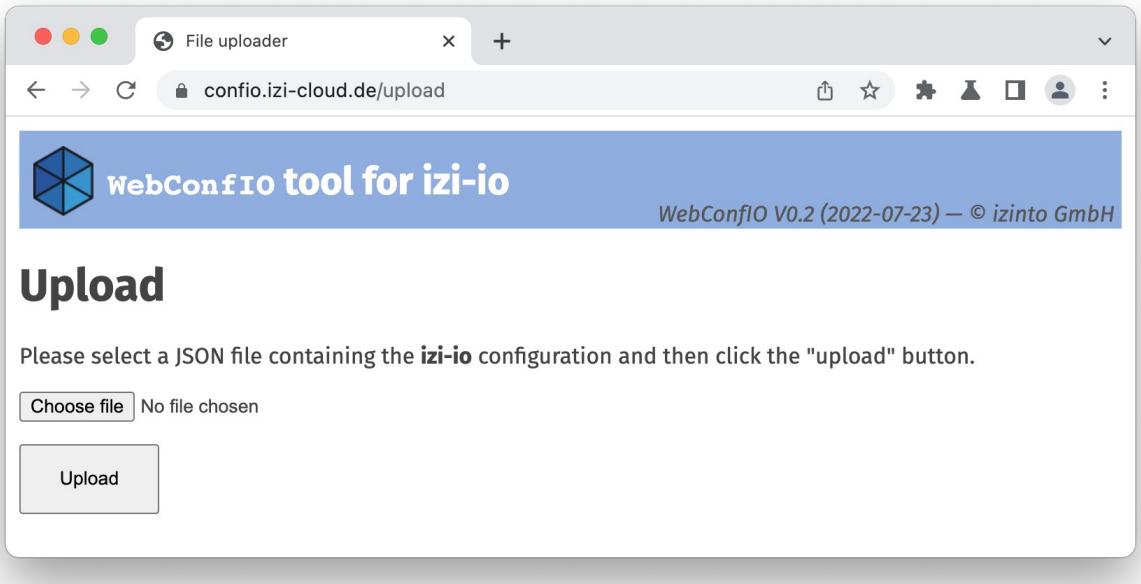

Abb. 2: Upload-Formular

Mit Klick auf den Button "Choose File" wird ein Dateiauswahldialog gezeigt, in dem die zu verarbeitende JSON-Konfigurationsdatei ausgewählt werden kann. Je nach Browser kann alternativ diese Datei auch einfach per Drag&Drop auf den Button gezogen werden.

Der Name der ausgewählten Datei wird zur Kontrolle anstelle des Textes "No file chosen" angezeigt. Ein Klick auf den Button "Upload" schließlich veranlasst das Hochladen der Datei und das Auflegen des Parametrierungsdialogs. Bei gravierenden Syntaxfehlern in der Datei wird stattdessen eine Fehlermeldung gezeigt.

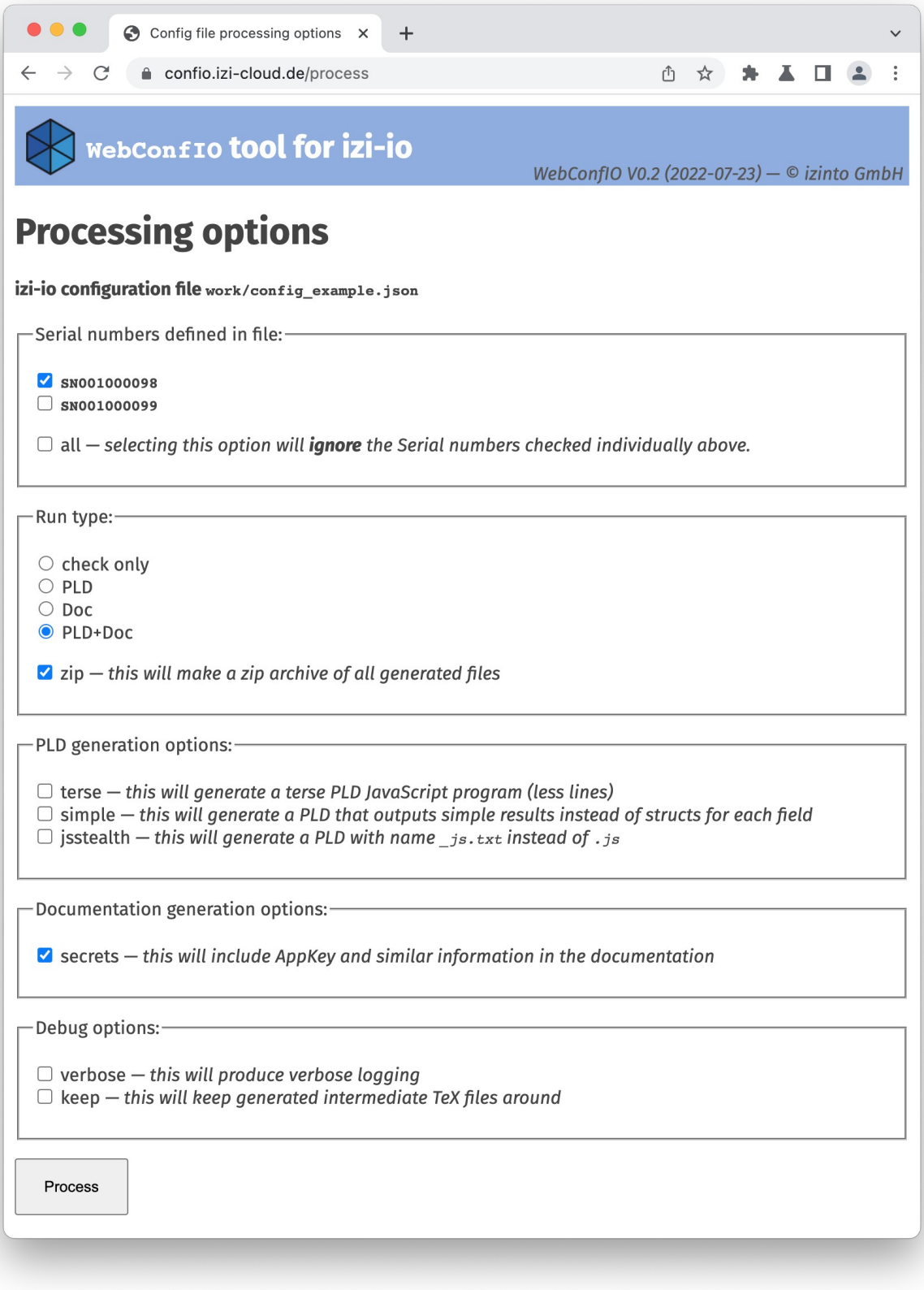

Abb. 3: Parametrierungsdialog

Der Dialog ist wie folgt aufgebaut:

- **Serial numbers:** eine Auflistung der in der JSON-Datei enthaltenen Seriennummern
  - Hier kann für jede Seriennummer festgelegt werden, ob der Payload Decoder bzw. die Dokumentation dazu generiert werden sollen.
  - Wenn "**all**" angekreuzt wird, werden – unabhängig von den Einzel-Einstellungen darüber – sämtliche Seriennummer bearbeitet.
- **Run type:** Festlegung, welche Artefakte das Tool erzeugen soll. Mögliche Optionen:
  - **check only:** Es wird nur die Datei eingelesen und analysiert. Bei Fehlern werden diese gemeldet.
  - **PLD:** Der Payload Decoder wird erzeugt.
  - **Doc:** Die Dokumentation wird erzeugt.
  - **PLD+Doc:** Sowohl Payload Decoder als auch Dokumentation werden erzeugt.
  - **zip:** Hier kann zusätzlich festgelegt werden, ob sämtliche erzeugten Artefakte gemeinsam mit der JSON-Quelldatei als ZIP-Archiv bereitgestellt werden sollen.
- PLD generation options:
  - **terse:** Bei Aktivierung dieser Option wird ein möglicht kompaktes JavaScript-Programm erzeugt. Anderenfalls enthält die Datei bei ansonstem identischem Inhalt mehr Whitespace und ist so besser lesbar.
  - **simple:** Es wird ein PLD erzeugt, der ein einfacheres Datenformat ausgibt. Anstelle einer Struktur mit den Keys `value`, `unit` und ggf. `error` wird für die übermittelten Werte direkt der Zahlenwert oder im Fehlerfall `null` erzeugt.
  - **jsstealth:** Um Probleme beim Versand der ZIP-Datei per eMail zu vermeiden, bekommen die PLD-Dateien anstelle des Dateinamens `x.js` den Namen `x_js.txt`.
- **Documentation generation options:**
  - **secrets:** Wenn aktiviert, werden die geheimen LoRaWAN-Schlüssel nicht in die Dokumentation übernommen.
- **Debug options:**

- **verbose:** Wenn aktiviert, werden mehr Informationen über den Ablauf der Verarbeitung protokolliert. Dies kann hilfreich sein, wenn die Konfiguration nicht fehlerfrei ist.
- **keep:** Für externe Nutzer nicht relevant.

Nach Einstellen der gewünschten Optionen wird durch Klick auf den Button "Process" das Tool gestartet. Im Schnitt werden pro generiertem PDF etwa 7 Sekunden benötigt, alle anderen Schritte benötigen keine nennenswerte Zeit.

Nach dem Lauf erscheint die Ergebnis-Seite.

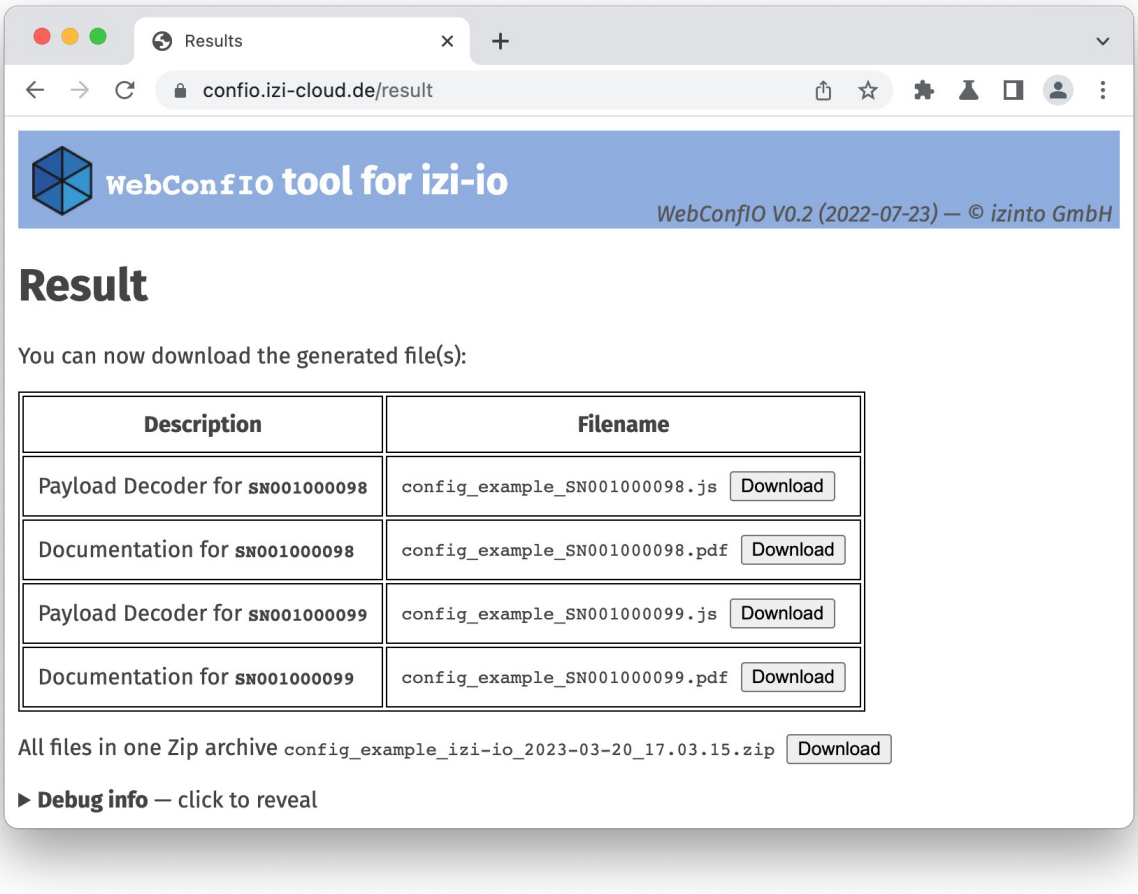

Abb. 4: Ergebnis-Seite

Diese zeigt in einer Tabelle sämtliche generierten Artefakte (JavaScript- und PDF-Dokumente), jeweils mit einem Download-Button. Falls aktiviert, wird auch das erzeugte ZIP-Archiv zum Download angeboten. Zusätzlich kann die Zeile "Debug Info" mit weiteren Informationen zum Lauf aufgeklappt werden.

4 Firmware-Varianten

Die **izi-io** ist in mit verschiedenen Firmware-Varianten erhältlich. Neben den serienmäßigen Varianten können auch kundenspezifische spezielle Implementierungen erstellt werden, die in diesem Dokument nicht behandelt werden.

Derzeit sind folgende Varianten lieferbar:

Firmware	**Bezeichner**	**Markierung hier im Dokument**
izi-io Basisfirmware	`izi-io`	`–keine–`
izi-grid Version 2 für Jean Müller PLMulti-II	`grid_plmulti_v2`	`*grid`
izi-grid Version 3 für Jean Müller PLMulti-II	`grid_plmulti_v3`	`*grid3`
izi-grid Version 4 für Jean Müller PLMulti-II	`grid_plmulti_v4`	`*grid4`
izi-grid Version 5 für Jean Müller PLMulti-II	`grid_plmulti_v5`	`*grid5`
izi-grid Version 6 für Jean Müller PLMulti-II	`grid_plmulti_v6`	`*grid6`

In der Konfigurationsdatei ist anzugeben, auf welche Firmware die Konfiguration bezogen werden soll. Davon abhängig stehen verschiedene vordefinierte Einstellungsparameter sowie Prozessvariablen zur Verfügung. Im folgenden Text wird jeweils darauf hingewiesen.

5 JSON-Konfigurationsdatei

Vorab: Eine Beispieldatei mit der Konfiguration für eine **izi-io /grid** befindet sich in Anhang . Bei Unklarheiten zu den beschriebenen Sachverhalten kann sie gegebenenfalls helfen, den Gesamtzusammenhang zu sehen.

5.1 Grundsätzliches zum JSON-Datenformat

Die Konfigurationsdatei wird als JSON mit etwas gelockerter Syntax verarbeitet: Das Skript zur Erzeugung des Payload Decoders akzeptiert JSON5, die izi-io-Firmware lediglich Standard-JSON mit zusätzlich erlaubten Kommata am Ende von Aufzählungen, d.h. vor schließenden geschweiften und eckigen Klammern. Standard-JSON-Syntax wird in jedem Fall akzeptiert.

JSON-Dateien sind lesbare Textdateien, die als ASCII-Datei oder als Unicode-Datei in UTF-8-Codierung vorliegen. Nicht-ASCII-Zeichen sind ausschließlich innerhalb von Strings erlaubt, sollten aber generell vermieden werden.

Als Anführungszeichen sind die normalen Anführungszeichen `""` (Unicode: Quotation Mark, Code U+0022, ASCII-Zeichen 34) zu verwenden, nicht die typographischen Anführungszeichen **„“** (Unicode: Double Low-9 Quotation Mark, Code U+201E bzw. Left Double Quotation Mark, Code U+201C). Bei Verwendung eines Texteditors, der für Programmtexte vorgesehen ist, werden die korrekten Anführungszeichen verwendet.

Zwischen den einzelnen JSON-Elementen kann beliebig Leerraum, sog. Whitespace, stehen. Dieser Leerraum besteht aus Leerzeichen (Unicode U+0020, ASCII 32), Tabulatorzeichen (U+0009, ASCII 9), Zeilenschaltungen (Carriage Return, CR, U+000D, ASCII 13 sowie[1] Linefeed / Line Tabulation, LF, U+000B, ASCII 11). Hiermit kann die Lesbarkeit der Datei erhöht werden, ohne sie semantisch zu ändern – etwa durch Einrückungen oder das Ausrichten der Values untereinander in derselben Position. Innerhalb von Anführungszeichen ist Whitespace Bestandteil des Strings und wird daher dort *nicht* ignoriert.

Die Parameter werden in JSON nach dem Key/Value-Prinzip notiert. Das bedeutet, ein Parametersatz besteht aus einer Parameterbezeichnung (engl. *key*, z.B. **`"Modbus_Baudrate"`**), immer als String mit Anführungszeichen, und dem zugeordneten Parameterwert (engl. *value*, z.B. `115200`, hier als Zahl). Bei sämtlichen angegebenen Keys ist die Groß-/Kleinschreibung zu beachten!

1 Dies bedeutet, dass JSON-Dateien sowohl nach Unix-Konvention (LF) als auch nach Windows-Konvention (CR+LF) akzeptiert werden.

Beispiel für ein Key-Value-Paar:

```
"Modbus_Baudrate": 115200,
```

Als erstes steht der Key in Anführungszeichen, gefolgt von einem Doppelpunkt und dem Value. Wenn nach einem Key/Value-Paar noch ein weiteres oder mehrere weitere Paare folgen, ist mit einem Komma abzuschließen. Das Komma selbst gehört nicht zum Key/Value-Paar, sondern dient als Trennzeichen zwischen solchen Paaren. Um die Bearbeitung der JSON-Dateien zu erleichtern, werden die in Standard-JSON nicht erlaubten Kommata vor abschließenden Klammern, d.h. nach dem letzten Paar, toleriert.

Es werden die folgenden Typen von Werten (Values) verarbeitet:

- Ganze Dezimalzahl (Integer): Diese wird direkt ausgeschrieben.
  `"Modbus_Baudrate": 115200`
- Fließkommazahl (Float): Diese wird mit einem Punkt als Dezimaltrennzeichen direkt ausgeschrieben. Exponentialschreibweise ist möglich.
  ```
  "PV_at_data_zero": -655.35,
  "RLine":           1.5e3
  ```
- Hexadezimalzahl (Hex): Diese wird in Anführungszeichen geschrieben.
  `"appeui": "82C4AAE7102219B7"`
  Achtung: Die hexadezimale Schreibweise ist ausschließlich bei solchen Parametern möglich (und dort auch zwingend einzuhalten), die in der Dokumentation entsprechend gekennzeichnet sind. Anders als in vielen Programmiersprachen wird hier kein Präfix „`0x`“ verwendet!
- Zeichenkette (String): Diese wird in Anführungszeichen geschrieben:
  `"Description": "Momentanwert Gesamtverzerrung Strom Einspeisung L2"`
- Binäre Parameter (Flags): Es wird entweder `true` oder `false` direkt ausgeschrieben, ohne Anführungszeichen:
  `"Priority": false`
  Achtung: Es gibt auch einige Parameter, die semantisch ebenfalls einen binären Schalter darstellen, aber vom Typ String sind und als erlaubte Values beispielsweise `"enabled"` und `"disabled"` haben. **Diese Values müssen wie jeder String in Anführungszeichen geschrieben werden.**
- JSON-Struktur: Der Wert besteht aus einer verschachtelten JSON-Struktur mit weiteren Key/Value-Paaren, welche in geschweifte Klammern gefasst ist. Die Paare in der Struktur sind wie üblich durch Kommata getrennt. Beispiel:
  ```
  "RequiredVersions": {
   "Hardware":   "1.1.0",
   "Kernel":     "1.1.0",
  ```

```
 "Firmware":   "1.3.0"
 }
```

Die Klammerung strukturiert die Hierarchieebenen und ist zwingend einzuhalten.

- JSON-Liste: Einem Key kann anstelle eines einzigen Values oder einer Struktur auch eine Liste mit mehreren Values zugeordnet werden. Diese Liste ist von eckigen Klammen umschlossen und die Values sind durch Kommata getrennt. Beispiel für eine Liste aus Zahlen:
  `"Numbers": [11, 23,99]`
  Anstelle eines Keys dient zur Identifikation der Listenelemente deren Index, der intern mit 0 beginnt, d.h. die Indizes der Liste im Beispiel gehen von 0 bis 2.

Listen und hierarchische Strukturen lassen sich auch kombinieren. Beispiel für eine Liste, die aus JSON-Strukturen besteht:

```
"Input_Config": [{
            "chanNo": 0,
            "RLine": 0,
            "Mode": "RTD_Pt1000",
            "Unit": "°C",
      },
      {
            "chanNo": 1,
            "Mode": "DIN_24V_PLC",
      },
      {
            "chanNo": 7,
            "Mode": "DIN_24V_PLC",
      },
]
```

Die blaugrau hervorgehobenen „überzähligen“ Kommata , sind wie beschrieben nur in JSON5 erlaubt.

Bevor die JSON-Datei mit einer izi-io zum Einsatz kommt, ist es empfehlenswert, die Datei auf korrekte Syntax zu überprüfen, damit die Parametrierung nicht mit einem Fehler abgebrochen wird.

Hinweis: Zum Editieren und Überprüfen der JSON-Datei empfehlen wir einen Texteditor mit Syntax-Highlighting und Fehlerprüfung für JSON. Beispielsweise ist unter Windows der freie Texteditor **Notepad++** hierfür geeignet, erhältlich unter: https://notepad-plus-plus.org. Eine Überprüfung auf korrekte Syntax ist z.B. mit Hilfe der Erweiterung „JSON Viewer“ für Notepad++ möglich. Die Erweiterung wird über das Menü „Erweiterungen → Plugin-Verwaltung“ installiert. Mit installierter „JSON Viewer“-Erweiterung kann eine

Überprüfung über das Menü „Erweiterungen → JSON Viewer → Show JSON viewer“ oder über die Tastenkombination ctrl+alt+shift+J gestartet werden.

Als Texteditor unter macOS empfehlen wir das Open-Source-Programm **SubEthaEdit**, kostenfrei erhältlich unter https://subethaedit.net sowie im App Store. Im Lieferumfang der Linux-Distributionen mit KDE ist **Kate** als ein geeigneter Editor enthalten, mit GNOME dementsprechend **gedit** oder zukünftig **GNOME Text Editor**. Alle genannten Editoren verfügen über Syntax Highlighting für JSON.

Unter Unix / Linux / macOS kann die Syntax mit folgendem Kommando geprüft werden, wenn Python installiert ist – was bei den meisten Systemen standardmäßig der Fall ist:

```
python -mjson.tool filename.json >/dev/null
```

Dabei wird auf die „klassische“, d.h. strenge JSON-Syntax geprüft. In diesem Fall werden Kommata nach Aufzählungen beanstandet. Wenn das Modul JSON5 installiert wurde (etwa mittels `pip install json5`), kann mit

```
python -mjson5.tool filename.json >/dev/null
```

entsprechend gemäß der hier gültigen JSON5-Syntax geprüft werden. Wenn das Kommando keine Ausgabe produziert, liegen keine Syntaxfehler vor.

5.2 Struktur der izi-io-Konfigurationsdatei

Auf der obersten Ebene hat die JSON Konfigurationsdatei für izi-io diese Form:

```
{
    "allDevices": { configuration0 },
    "serno1":     { configuration1 },
    "serno2":     { configuration2 },
    …
}
```

serno1 aus dem Beispiel *steht für eine Seriennummer, welche bei der izi-io Gerätereihe beispielsweise* `SN001000040` *lauten kann.* *configuration1* *steht für* eine JSON-*Struktur*, also eine Auflistung beliebig vieler Key/Value-Paare. Jede Konfiguration selbst wiederum ist in Abschnitte strukturiert, die im folgenden beschrieben werden.

Jedem individuellen Gerät wird ein anhand seiner Seriennummer ein eigener Parametersatz zugewiesen. Der Eintrag **`allDevices`** hat eine besondere Funktion. Dort können Defaultwerte für sämtliche Geräte abgelegt werden. Bei der Verarbeitung eines Eintrags für eine spezifische Seriennummer wird immer dann, wenn für einen Parameter kein gerätespezifischer Wert vorliegt, der Standardwert aus **`allDevices`** verwendet.

Beispiele:

- Die seriennummernspezifischen Einträge enthalten die LoRaWAN-Schlüssel (s.u.). Dieser Eintrag fehlt in der Defaultkonfiguration.
- In der Defaultkonfiguration ist ein Grenzwert für das zulässige Spannungsband definiert, der für alle Geräte gültig ist. In einigen seriennummernspezifischen Einträgen wird ein einsatzbedingt abweichender Wert konfiguriert, der den Defaultwert überschreibt.

Auf diese Weise kann an einer zentralen Stelle in der JSON-Datei z.B. die Messkonfiguration für alle Geräte einer Installation vorgegeben werden, während etwa die LoRaWAN-Zugangsdaten für jedes Gerät getrennt voneinander parametrierbar sind.

Auf der zweiten Ebene (Inhalt von *configurationx*) ist die JSON-Datei in Abschnitte zu den einzelnen Systembestandteilen unterteilt, welche dann jeweils eine JSON-Struktur mit den Parametern zu diesem Abschnitt beinhalten. Die Abschnitte werden unter den folgenden Keys abgelegt:

LoRaWAN_Param	LoRaWAN-Netzwerkparameter, siehe Abschnitt 5.4
LoRaWAN_Keys	LoRaWAN-Netzwerkschlüssel, siehe Abschnitt 5.5
PL_Multi_Param *grid	Parameter für die Modbus-Kommunikation mit einem PLMulti-II Messumformer, siehe Abschnitt 5.6 Gültig ab Firmwarevariante grid_plmulti_v2
ACgrid_Param *grid	Parameter für die Überwachung elektrisches Netz, siehe Abschnitt 5.7 Gültig ab Firmwarevariante grid_plmulti_v2
IO_Edgeproc_Param *grid4	Parameter für Optionen zum Edge Processing, siehe Abschnitte 5.8 und 5.9. Gültig ab Firmwarevariante grid_plmulti_v4
Input_Config	Konfiguration der analogen / digitalen Messeingänge, siehe Abschnitt 5.8
Bit_Items	Definition der Bitfelder im LoRaWAN-Telegramm, siehe Abschnitt 5.12
LoRaWAN_Telegrams	Definition der LoRaWAN-Telegramme, siehe Abschnitt 5.15

Im Folgenden werden jeweils zu den Parametern der erwartete Datentyp (Integer, Float, Hex, String, Flag) sowie die Beschreibung angegeben.

Hinweis: Parameter, die in einer Konfigurationsdatei *nicht* gesetzt werden, behalten den Wert aus der vorhergehenden Konfiguration, d.h. sie werden nicht überschrieben. Initial

sind sie mit den unten angegebenen Werkseinstellungen belegt. Im Folgenden ist die Werkseinstellung eines Parameters mit einem Dreieck gekennzeichnet

5.3 Kopf der Konfiguration

Direkt auf oberster Ebene innerhalb der Konfiguration werden folgende Einträge erkannt:

Name [String]
Gerätename; dient ausschließlich der Dokumentation.

Description [String]
Beschreibung; dient ausschließlich der Dokumentation.

Firmwaretype [String]
Typ der installierten Firmware, für die diese Konfigurationsdatei bestimmt ist., vgl. Abschnitt 4. Die Konfigurationsdatei darf nur auf Geräte aufgespielt werden, auf denen auch dieser Firmwaretyp installiert ist. Beispiel: `grid_plmulti_v2` ist die (minimale) Firmware der **izi-io /grid** für den Anschluss an ein PLMulti-II.

RequiredVersions eine Struktur, die in **Hardware**, **Kernel** und **Firmware** als Key aufgeteilt ist. Zu jedem Key wird die entsprechende Version [String] angegeben, die im Zielsystem *mindestens* vorhanden sein muss.[2]

5.4 LoRaWAN-Parameter

Unter dem Key **LoRaWAN_Param** werden die in diesem Abschnitt vorgestellten Parameter erkannt.

Mode [String]
LoRaWAN-Region. Aktuell möglich:
- ▶ **EU868**: für 868MHz Band, Region EU
- • **off**: LoRaWAN-Modem abgeschaltet

Class [String]
LoRaWAN-Klasse. Aktuell möglich:
- ▶ **A**

2 Details zur Syntax der Versionsnummern sind noch TBD.

PrioMode [String]
Modus für priorisierte Telegramme:
• **repeat**: Telegramm wird wiederholt ausgesendet, die Anzahl ist in **confRetrNr** parametrierbar.
▸ **Confirmed**: Telegramm wird wird als confirmed message ausgesendet und vom Netzwerk bestätigt. Die Aussendung wird wiederholt, falls die Bestätigung ausbleibt; die maximale Anzahl Versuche ist auf **confRetrNr** begrenzt.
Vgl. Telegramm-Parameter **Priority** und **PrioOnEventsOnly** unten in Abschnitt 5.15.1.

JoinMode [String]
Aktivierungsmodus für den Beitritt zum Netzwerk:
▸ **OTAA**: over the air activation
• **ABP**: activation bei personalization
Abhängig vom Aktivierungsmodus müssen die jeweils erforderlichen LoRaWAN-Schlüssel in der JSON-Datei vorhanden sein bzw. sich im Speicher des LoRaWAN-Moduls der izi-io befinden, siehe hierzu Abschnitt 5.5.

OTAAreboot [String]
Verhalten im OTAA-Modus bei einem Neustart des izi-io-Gerätes nach einem Spannungsausfall oder Reset:
▸ **forcerejoin**: es wird immer ein OTAA Join durchgeführt.
• **dontforce**: wenn im Gerät gültige Session Keys vorhanden sind, unterbleibt ein neuer OTAA Join und die vorhandenen Keys werden weiter verwendet.

DR [String]
Initiale Uplink-Datenrate (Senderichtung) nach einem Join oder der Neuinitialisierung der Funkschnittstelle. Die Datenrate kann bei aktivierter automatischer Anpassung (siehe **AdaptDR**) seitens des Netzwerkservers geändert werden. Gültige Werte hängen von den Regionaleinstellungen ab.
Mögliche Werte für EU868:
• **SF7/250kHz**
• **SF7/125kHz**
• **SF8/125kHz**

- **SF9/125kHz**
- ▶ **SF10/125kHz**
- **SF11/125kHz**
- **SF12/125kHz**

DR_RX2 [String]
Initiale Datenrate für das Downlink-Zeitfenster (Empfangsrichtung). Dieser Parameter muss mit dem entsprechenden Parameter auf der Serverseite übereinstimmen. Gültige Werte hängen von den Regionaleinstellungen ab. Mögliche Werte für EU868:

- **SF7/250kHz**
- **SF7/125kHz**
- **SF8/125kHz**
- ▶ **SF9/125kHz**
- **SF10/125kHz**
- **SF11/125kHz**
- **SF12/125kHz**

AdaptDR [String]
Schalter für den ADR-Modus (adaptive data rate):

▶ **enabled**: Die Netzwerkparameter werden vom Server automatisch angepasst.

- **disabled**: Es werden stets die parametrierten Netzwerkparameter beibehalten.

AutomReply [String]
Freigabe zur Aussendung eines Antworttelegramms auf ein confirmed downlink Telegramm vom Server oder auf ein Telegramm vom Server, bei dem das Frame pending Bit gesetzt ist:

▶ **enabled**: Antworttelegramm wird an Server gesendet.

- **disabled**: keine Antwort.

t_linkchk [Integer]
Zeitintervall für den link check Prozess (0 bis 65535 s). Wenn diese Zeit in Sekunden abgelaufen ist, wird beim nächsten ausgesendeten Uplink-Telegramm das Kommando *Link Check* mitgesendet und der Netzwerkserver antwortet mit einem

Downlink-Paket zur Bestätigung.

- Default-Wert: 0 (Funktion deaktiviert).

t_inhibit [Integer]
Globale Sperrzeit (0 bis 32767 s) nach der Aussendung eines Telegramms. Die Sperrzeit in Sekunden startet nach jedem gesendeten Telegramm. Solange die Sperrzeit läuft, werden Sendeanforderungen für weitere Telegramme zurückgehalten. Diese Sperrzeit bestimmt die schnellste Folge, in der Telegramme ausgesendet werden können. Siehe auch Abschnitt 5.14.

- Default-Wert: 60

t_wait_err [Integer]
Wartezeit (30 bis 32767 s) nach einem fehlgeschlagenen Sendeversuch, bevor das Telegramm erneut gesendet wird.

- Default-Wert: 60

Err_retr [Integer]
maximale Anzahl von erfolglosen Sendeversuchen im Intervall **t_wait_err**, bevor die Wiederholrate auf den Sicherheitswert von einem Sendeversuch pro Stunde reduziert wird (1 bis 255).

- Default-Wert: 255

Tx_power [Integer]
Initiale Sendeleistung nach einem Join oder der Neuinitialisierung der Funkschnittstelle. Die Sendeleistung kann bei aktivierter automatischer Anpassung (siehe **AdaptDR**) vom Netzwerkserver geändert werden. Gültige Werte hängen von den Regionaleinstellungen ab. In der Region EU868 sind die Werte 1 bis 5 zulässig.

- Default-Wert: 1

confRetrNr [Integer]
Anzahl der Wiederholungen bei confirmed Telegrammen, siehe auch **PrioMode** (1 bis 255).

- Default-Wert: 4

rxdelay1 [Integer]
Zeitverzögerung in Millisekunden, nach der sich im Anschluss an ein Sendetelegramm das RX1-Fenster öffnet. Während des RX1-Fensters kann das Gerät Downlinktelegramme vom Netzwerkserver empfangen. Dieser Parameter muss mit dem entsprechenden Parameter auf der Serverseite übereinstimmen.
- Default-Wert: 1000

rx2_freq [Integer]
Empfangsfrequenz für das RX2-Fenster in Hz (863000000 bis 870000000). Dieser Parameter muss mit dem entsprechenden Parameter auf der Serverseite übereinstimmen.
- Default-Wert: 869525000

synchWord [Integer]
Verwendetes Synchronization Word im LoRa-Telegramm (0 bis 255). Dieser Parameter muss mit dem entsprechenden Parameter auf der Serverseite übereinstimmen.
- Üblich ist bei den meisten Netzwerkservern der Wert 52.

Wichtig: die Serverseite muss auf die LoRaWAN-Protokollversion 1.0.2 parametriert werden. Das bei der **izi-io** verwendete LoRaWAN-Modul (Microchip RN2483) ist nicht abwärtskompatibel zu höheren Protokollversionen und kann unvorhergesehen auf nicht unterstützte Netzwerkmanagementbefehle vom Server reagieren.

Anhang A enthält eine beispielhafte Konfigurationsdatei, in der die Parameter mit den Defaulteinstellungen gezeigt werden.

5.5 LoRaWAN-Zugangsdaten

Unter dem Key **LoRaWAN_Keys** werden, abhängig vom Aktivierungsmodus, die LoRaWAN-Deployment-Parameter des Geräts angegeben. Es muss, passend zu jeweiligen Aktivierungsmodus, immer ein vollständiger Parametersatz zur Verfügung stehen. Die einzelnen Parameter können auch aus Default- und Individualangaben kombiniert werden, wenn beispielsweise alle Geräte eine gemeinsame AppEUI haben.

Die Zugangsdaten müssen als Hexadezimalzahl (niederwertigste Ziffer ganz rechts) angegeben werden.

Bei OTAA sind dies:

deveui [Hex] 8 Byte Device EUI (DevEUI)

appeui [Hex] 8 Byte Application EUI (AppEUI bzw. JoinEUI)

appkey [Hex] 16 Byte Application Key (AppKey)

Bei ABP sind dies:

nwkskey [Hex] 16 Byte Network Session Key (NwkSKey)

appskey [Hex] 16 Byte Application Session Key (AppSKey)

devaddr [Hex] 4 Byte Device Address (DevAddr)

5.6 Parameter PLMulti-Messumformer `*grid`

Unter dem Key **PL_Multi_Param** können, wenn die izi-io Firmware `grid_plmulti_v2` (oder eine spätere Version) installiert ist, die Parameter für die Modbus-Kommunikation mit einem Energiemessumformer PLMulti-II von Jean Müller spezifiziert werden.

Abgesehen von **Modbus_serialMode** sind alle Modbus-Kommunikationsparameter auch im PLMulti-Gerät enthalten. Siehe hierzu Abschnitt 7.2.4 der PL-Multi Betriebsanleitung (Dokument „PLMulti-II V1.260 BA-E040 | 14839c“ der Jean Müller GmbH). Damit eine fehlerfreie Kommunikation zustande kommt, müssen diese Parameter in beiden Geräten übereinstimmen.

Wichtig: Im PLMulti-Gerät sollte die neueste Firmware des Herstellers installiert sein. Mindestvoraussetzung für eine erfolgreiche Kommunikation ist die Version 1.26x.

Die mit einem Dreieck gekennzeichneten Default-Parameterwerte (z.B. „▶ *none“)* sind in der Firmware der izi-io hinterlegt und werden gültig, falls bei der Erstinbetriebnahme der Abschnitt **PL_Multi_Param** in der `config.json` nicht vorhanden sein sollte. Diese Parameter entsprechen den Werkseinstellungen des PLMulti-II und brauchen normalerweise nicht geändert zu werden. Auch hier enthält die Beispieldatei in Anhang A die jeweiligen Defaultwerte.

Modbus-Parameter:

Float32_Format [String]
Bytefolge für Float-Zahlen. Die Übertragungsreihenfolge der vier Bytes, aus denen eine Floatzahl besteht, kann hier konfiguriert werden, siehe hierzu auch die Bedienungsanleitung des PL- Multi Gerätes, Abschnitt 7.2.4.

- ▶ **CDAB**,
- **DCBA**,

- **ABCD**
- **BADC**

Modbus_parity [String]
Parität der seriellen Modbus-Schnittstelle:
▶ **none**: kein Paritätsbit
- **even**: gerade Parität
- **odd**: ungerade Parität.

Modbus_serialMode [String]
Modus der izi-io Hardwareschnittstelle. Das PLMulti-II unterstützt nur RS485; die anderen Modi sind eventuell sinnvoll, wenn zwischen PLMulti und izi-io ein Medienkonverter geschaltet ist, beispielsweise um eine größere Entfernung zu überbrücken.
▶ **RS485**: Verbindung über RS485 (Bus-Signale A, B).
- **RS422**: Verbindung über RS522 (Bus-Signale A, B, X, Y).
- **RS232**: Verbindung über RS232 (Punkt-zu-Punkt-Signale Tx, Rx)

Modbus_Baudrate [Integer]
Baudrate der seriellen Modbus-Schnittstelle (1200 bis 10500000)
▶ Default-Wert: 115200.

Modbus_stopbits [Integer]
Stopbits der seriellen Modbus-Schnittstelle (1 oder 2)
▶ Default-Wert: 1

PL_Multi_SlaveAdr [Integer]
Modbus-Adresse des PLMulti-Geräts (1 bis 247)
▶ Default-Wert: 10.

RTC_syncMode [String] *grid4
Modus für Uhrenbetrieb:
- **noSync**: Es wird keine Zeitinformation eingelesen.

▶ **SyncFromPLMulti**: Die im PLMulti vorhandene Echtzeituhr wird als Unix-Timestamp (Anzahl Sekunden seit 01.01.1970 UTC) ausgelesen, die Zeitinformation ist dann in der izi-io nutzbar.

Bei Nutzung dieser Option ist sicherzustellen, dass die Uhr im PLMulti korrekt gestellt ist.

5.7 Parameter ONS-Option (Überwachung elektrisches Netz)

Unter dem Key **ACgrid_Param** können bei Verwendung der entsprechenden Firmware (ab `grid_plmulti_v2`) die Grenzwerte parametriert werden, die zur Überwachung der Messgrößen eines elektrischen Netzes vorgesehen sind. Bei Verletzung der Grenzwerte wird der Trigger der entsprechenden Prozessvariablen gesetzt, siehe auch Abschnitt 5.14.

In der JSON-Datei anzugeben sind die numerischen Werte in der angegebenen Maßeinheit, ohne die Maßeinheit, Beispiel:

```
"U_Lim_L": 207,
"P_Lim_H": 100
```

Hier wird der untere Grenzwert für die Phasenspannungen auf 207 V gesetzt, der obere Grenzwert für das Leistungsfenster der Phasen auf 100 kW.

mean_interval [Integer] `*grid4`
Mittelungsintervall für Energiemessgrößen in Sekunden, maximal 86400 s = 24 h.
▶ Default-Wert: 0 (Bildung der Mittelungsgrößen deaktiviert).

mean_intrv_offset [Integer] `*grid4`
Zeitoffset zum Mittelungsintervall für Energiemessgrößen in Sekunden, maximal 86400 s = 24 h. Wird auf den Unix-Timestamp addiert. Auf diese Weise können Mittelungsintervalle nicht nur zur vollen Stunde / Viertelstunde / etc. realisiert werden, sondern auch zu Zeitpunkten, die um den angegebenen Zeitraum hinter diese Zeitmarken verschoben sind.
▶ Default-Wert: 0

U_Lim_L [Float]
unterer Grenzwert Spannungsband (0 bis 400 V). Wenn eine der Phasenspannungen (L–N) den Grenzwert unterschreitet, wird ein Trigger für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

U_Lim_H [Float]
oberer Grenzwert Spannungsband (0 bis 400 V). Wenn eine der Phasenspannungen (L–N) den Grenzwert überschreitet, wird

ein Trigger für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

U_Lim_sym [Float]
Grenzwert Symmetrie Spannung (0 bis 400 V). Wenn die Differenz zwischen einer der Phasenspannungen (L–N) zum Median aller drei Phasenspannungen den Grenzwert überschreitet, wird ein Trigger für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

I_Lim_L [Float]
unterer Grenzwert Phasenstrom (0 bis 1250 A), Funktion sinngemäß entsprechend **U_Lim_L**.
▶ Default-Wert: 0

I_Lim_H [Float]
oberer Grenzwert Phasenstrom (0 bis 1250 A), Funktion sinngemäß entsprechend **U_Lim_H**.
▶ Default-Wert: 0

I_Lim_sym [Float]
Grenzwert Symmetrie Phasenströme (0 bis 1250 A), Funktion sinngemäß entsprechend **U_Lim_sym**.
▶ Default-Wert: 0

P_Lim_L [Float]
unterer Grenzwert Wirkleistung (0 bis 330 kW). Wenn die Leistung einer Phase den Grenzwert unterschreitet, wird ein Trigger für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

P_Lim_H [Float]
oberer Grenzwert Wirkleistung (0 bis 330 kW). Wenn die Leistung einer Phase den Grenzwert überschreitet, wird ein Trigger für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

P_Lim_sym [Float]
Grenzwert Symmetrie Wirkleistung (0 bis 330 kW). Wenn die Differenz der Leistung einer Phase zum Median der Leistungen aller drei Phasen den Grenzwert überschreitet, wird ein Trigger

für diese Prozessvariable gesetzt.
▶ Default-Wert: 0

THD_U_Lim_H [Float]
Grenzwert THD Spannung (0 bis 100 %). Wenn der Verzerrungswert der Spannung (Total Harmonic Distortion) einer Phase (L–N) diesen Grenzwert überschreitet, wird ein Trigger für diese Prozessvariable gesetzt. Diese Funktion steht nur zur Verfügung, wenn der PLMulti-II Messumformer auch mit der Power Quality Option ausgestattet ist.
▶ Default-Wert: 0

THD_I_Lim_H [Float]
Grenzwert THD für den Strom einer Phase (0 bis 100 %), Funktion sinngemäß entsprechend **THD_U_Lim_H**.
▶ Default-Wert: 0

Bei Verwendung der Standard-JSON-Konfigurationsdatei löst ein gesetzter Trigger die Aussendung der folgenden Telegramme aus; siehe auch Abschnitt 5.15:

- **U_Lim_L, U_Lim_H, U_Lim_sym, I_Lim_L, I_Lim_H, I_Lim_sym**:
  Telegramm „Typ B“, Portnummer 21 mit den Momentanwerten der Phasenspannungen und -Ströme.
- **P_Lim_L, P_Lim_H, P_Lim_sym**:
  Telegramm „Typ C“, Portnummer 22 mit den Momentanwerten der Phasenleistungen.
- **THD_U_Lim_H, THD_I_Lim_H**:
  Telegramm „Typ D“, Portnummer 23 mit den Momentanwerten der Spannungs- und Strom-Verzerrungen.

Diese Zuordnung zu den Telegrammen ist nicht festgelegt und kann in den Telegramm-Definitionen (siehe Abschnitt 4.15) geändert werden.

5.8 Parameter Loopcheck-Option: Zyklische Prüfungen *grid4

Unter dem Key **IO_Edgeproc_Param** kann bei Verwendung der entsprechenden Firmware (ab `grid_plmulti_v4`) die Loopcheck-Option parametriert werden. Diese Funktion startet zyklisch einen Prüfablauf, mit dem die Funktion einer extern angeschlossenen Signalquelle (z.B. eine Anlagensteuerung) und der Verbindung zur izi-io überprüft wird.

Nach Abschluss der Prüfung wird der Trigger der entsprechenden Prozessvariablen gesetzt, siehe auch Abschnitt 5.14.

Loopchk_function [String]
Gibt an, ob die Option aktiv sein soll:
- **enabled**: Option ist aktiv.
- ▶ **disabled**: Option ist nicht aktiv.

Loopchk_TestAlarm [String]
Gibt an, ob die während einer Prüfung auftretenden Alarme als LoRaWAN-Telegramme ausgesendet werden sollen:
- ▶ **sendTestAlarms**: Alarme werden ausgesendet.
- **muteAlarms**: Alarme werden unterdrückt.

Loopchk_timer [String]
Gibt an, ob der Zeitgeber uhrzeitsynchron betrieben werden soll:
- ▶ **FreeRunning**: Zeitgeber läuft frei (Intervalle beginnen mit Systemstart).
- **Clocksynced**: Zeitgeber läuft uhrzeitsynchron (Voraussetzung hierzu ist eine verfügbare Uhrzeit, vgl. auch Parameter **RTC_syncMode** in Abschnitt 5.6.

Loopchk_Interval [Integer]
Intervall für Prüfzyklus in Sekunden, erlaubter Bereich 60 bis 100000000 s = ca. 3 Jahre.
▶ Default-Wert: 0 (kein Loopcheck)

Loopchk_IntrvOffs [Integer]
Offset zum Intervall für Prüfzyklus in Sekunden, erlaubter Bereich 0 bis 100000000 s = ca. 3 a. Das Offset wird zum Unix-Timestamp hinzuaddiert. Auf diese Weise lässt sich der Zeitpunkt der Prüfung auf feste Uhrzeiten oder Wochentage festlegen. Nur wirksam im uhrzeitsynchronen Modus.

Loopchk_timeout [Integer]
Timeout für Prüfablauf in Sekunden, erlaubter Bereich 1 bis 600 s = 10 min. Wenn der Prüfablauf nicht in der angegebenen

Zeit durch Einlesen der geforderten Signale abgeschlossen werden konnte, gilt die Prüfung als nicht erfolgreich.

5.9 Parameter Betriebsstundenzähler-Option `*grid4`

Ebenfalls unter dem Key **IO_Edgeproc_Param** kann bei Verwendung der entsprechenden Firmware (ab `grid_plmulti_v4`) die Betriebsstundenzähler-Option parametriert werden, mit der die Einschaltzeiten von maximal zwei Geräten protokolliert werden kann. Wichtig ist, dass die Struktur mit dem Key **IO_Edgeproc_Param** nur einmal im JSON enthalten ist, d.h. die Parameter für diese Option und für die Loopcheck-Option werden in derselben Struktur gesetzt[3].

OpTime_cntr [String]
Gibt an, ob die Option aktiv sein soll:
- **enabled**: Option ist aktiv.
- ▶ **disabled**: Option ist nicht aktiv.

unit1_opsignal [String]
Gibt die Quelle für das Einschalt-Signal des Betriebsstundenzählers 1 an:
- ▶ **from_DI**: Digital-Eingang IO0.
- **from_AI**: Analog-Eingang IO2.

unit2_opsignal [String]
Gibt die Quelle für das Einschalt-Signal des Betriebsstundenzählers 2 an:
- ▶ **from_DI**: Digital-Eingang IO3.
- **from_AI**: Analog-Eingang IO5.

UnitMon_dmd_thr [float]
Definiert die Schaltschwelle für den Analog-Eingang, ab der ein Betrieb erkannt wird. Der Wert und der gültige Bereich sind entsprechend dem zugeordneten Analogeingang skaliert. Wenn der Analogeingang z.B. auf 0..10 bar skaliert ist, liegt der zulässige Wertebereich von **UnitMon_dmd_thr** bei 0..10.

3 Unabhängig davon ist es dennoch möglich, in der JSON-Gesamtstruktur den Default-Mechanismus zu nutzen (vgl. Abschnitt 5.2) – es dürfen nur nicht mehrere identische Keys innerhalb derselben Struktur nebeneinander stehen.

UnitMn_blnkTimOn [Float]
Ausblendzeit Betriebsmittelstart. Direkt nach dem Start eines Betriebsmittels wird für die Ausblendzeit keine Min/Max-Registrierung durchgeführt. Erlaubter Bereich 0 bis 600 Sekunden.

UnitMn_blnkTimOff [Float]
Ausblendzeit Betriebsmittelstop. In der parametrierten Zeit unmittelbar vor der Abschaltung eines Betriebsmittels wird für die Ausblendzeit keine Min/Max-Registrierung durchgeführt. Erlaubter Bereich 0 bis 600 Sekunden.

5.10 Konfiguration der Eingänge IO0 bis IO7

Unter dem Key **Input_Config** werden die analogen / digitalen Eingänge konfiguriert. Messwerte von Spannungen / Strömen / Widerstandswerten der analogen Eingänge IO0 bis IO7 können mit einer linearen Abbildung flexibel den Prozessvariablenwerten `Analog_In_IO0` bis `Analog_In_IO7`, welche sie repräsentieren sollen, zugeordnet werden. Digitale Eingangswerte erscheinen als Bit in der Bitfeld-Prozessvariablen `IOStatus`.

Die Parametrierung der Eingänge IO0..7 geschieht in Form einer JSON-Liste, deren Elemente Strukturen mit dem Parametersatz für jeweils einen Eingang sind, die aus folgenden Keys bestehen:

chanNo [Integer]
Kanalnummer (bei der **izi-io 4840** von 0 bis 7), auf die sich dieser Parametersatz bezieht.

analog_range_beg [Float]
analoger Eingangswert (V, mA, oder Ω), der dem unteren Messbereichsende entspricht. Dieser Wert ist dem Prozessvariablenwert **PV_range_beg** zugeordnet.

analog_range_end [Float]
analoger Eingangswert (V, mA, oder Ω), der dem oberen Messbereichsende entspricht. Dieser Wert ist dem Prozessvariablenwert **PV_range_end** zugeordnet.

PV_range_beg [Float]
Prozessvariablenwert am unteren Messbereichsende.

PV_range_end [Float]
Prozessvariablenwert am unteren Messbereichsende.

RLine [Float]
In den Widerstands- und Temperaturmessbereichen kann optional der Leitungswiderstand (in Ω) der Zuleitungen zum Messwertgeber parametriert werden. Dieser Leitungswiderstand wird vom gemessenen Widerstandswert subtrahiert. Im Strommessbereich 4..20 mA wird hier der extern verbaute Bürdenwiderstand in Ω angegeben. Üblich ist die Verwendung eines 20-Ω-Bürdenwiderstands.

Mode [String]
Betriebsmodus des Eingangs, siehe unten. Es sind folgende Werte möglich:

- **off** Eingang ist inaktiv
- **DIN_24V_PLC** Digitaleingang mit 24-Volt-Pegel (SPS-Standard): Der Eingang zieht einen konstanten Strom von 2,5 mA gegen GND und hat Schmitt-Trigger-Verhalten. Schaltschwelle H-Pegel bei U > 11 V. Schaltschwelle L-Pegel bei U < 5 V.
- **DIN_5V_logic** Digitaleingang mit 5-Volt-Pegel (TTL-Standard): der Eingang hat einen Eingangswiderstand von 100 kΩ und Schmitt-Trigger-Verhalten. Schaltschwelle H-Pegel bei U > 2,4 V. Schaltschwelle L-Pegel bei U < 0,8 V.
- **AIN_0.55V** Analogeingang 0..0,55 V, > 10 MΩ Eingangswiderstand.
- **AIN_3.3V** Analogeingang 0..3,3 V, > 10 MΩ Eingangswiderstand.
- **AIN_11V** Analogeingang 0..11 V, 100 kΩ Eingangswiderstand.
- **AIN_30V** Analogeingang 0..30 V, 100 kΩ Eingangswiderstand.
- **Resistor_1500Ohm** Widerstandsmessung 0..1500 Ω, Messstrom maximal 0,44 mA, Spannung maximal 3,3 V.
- **Resistor_70kOhm** Widerstandsmessung 0..70 kΩ, Messstrom maximal 0,44 mA, Spannung maximal 3,3 V.
- **RTD_PT1000** Temperaturmessung mit Pt1000-Fühler, Messstrom maximal 0,44 mA, Spannung maximal 3,3 V.

- **RTD_Ni1000_TK5000** Temperaturmessung mit Ni1000-Fühler (TK5000), Messstrom maximal 0,44 mA, Spannung maximal 3,3 V.
- **RTD_Ni1000_TK6180** Temperaturmessung mit Ni1000-Fühler (TK6180), Messstrom maximal 0,44 mA, Spannung maximal 3,3 V.
- **AIN_4_20mA** Analogeingang Stromschleife 4..20 mA; in diesem Modus muss ein externer Bürdenwiderstand von 20 Ω zwischen den Eingang und GND geschaltet werden.
- **DIN_24V_PLC_inv** Digitaleingang mit 24-Volt-Pegel (SPS-Standard), invertiert
- **DIN_5V_logic_inv** Digitaleingang mit 5-Volt-Pegel (TTL-Standard), invertiert

Beispiel für die Eingangskonfiguration: Angeschlossen ist ein Druckgeber mit dem Messbereich -1..3 bar, der einen Stromausgang 4..20 mA hat. Dann sind in der JSON-Datei die Parameter für die Zuordnung folgendermaßen zu setzen:

```
"Mode":              "AIN_4_20mA",
"RLine":             20,
"analog_range_beg":   4,
"analog_range_end":  20,
"PV_range_beg":      -1,
"PV_range_end":       3
```

5.11 Konfiguration der Ausgänge

Unter dem Key **Output_Config** werden die Ausgänge konfiguriert. Dieser Abschnitt ist in der derzeitigen Version der Standard-Firmware ohne Funktion.

5.12 Prozessvariablen

Prozessvariablen (PV) enthalten im laufenden Anwendungsprogramm Werte wie die gemessenen Eingangssignale, von extern übertragene Messwerte oder auch intern errechnete Werte. Sie stehen zur Verfügung, um einerseits den Versand von LoRaWAN-Telegrammen zu triggern und um andererseits daraus die Felder von LoRaWAN-Telegrammen zu bilden, so dass die Werte an nachgeordnete Systeme übermittelt werden.

Folgende Prozessvariablen stehen zur Verfügung:

Kat.	PV-Name	Typ	Verfügb.	Erläuterung
Status-Bitfelder				s. Abschnitt 5.13

Kat.	PV-Name	Typ	Verfügb.	Erläuterung
	SysStatus_Tel0	8Bit		
	SysStatus_Tel123	8Bit		
	IOStatus	8Bit		
	IO_Loopcheck_Stat	8Bit	*grid4	s. Abschnitt 5.8
Eingangssignale				s. Abschnitt 5.10
	Analog_In_IO0	Float32		Prozessvariable Analogeingang IO0
	Analog_In_IO1	Float32		Prozessvariable Analogeingang IO1
	Analog_In_IO2	Float32		Prozessvariable Analogeingang IO2
	Analog_In_IO3	Float32		Prozessvariable Analogeingang IO3
	Analog_In_IO4	Float32		Prozessvariable Analogeingang IO4
	Analog_In_IO5	Float32		Prozessvariable Analogeingang IO5
	Analog_In_IO6	Float32		Prozessvariable Analogeingang IO6
	Analog_In_IO7	Float32		Prozessvariable Analogeingang IO7
ONS-Option			*grid	s. Abschnitt 5.7
	U_L1_m	Float32		Spannung L1—N Mittelwert
	U_L2_m	Float32		Spannung L2—N Mittelwert
	U_L3_m	Float32		Spannung L3—N Mittelwert
	I_feed_L1_m	Float32		Strom Einspeisung L1 Mittelwert
	I_feed_L2_m	Float32		Strom Einspeisung L2 Mittelwert
	I_feed_L3_m	Float32		Strom Einspeisung L3 Mittelwert
	P_feed_L1_m	Float32		Wirkleistung Einspeisung L1 Mittelwert
	P_feed_L2_m	Float32		Wirkleistung Einspeisung L2 Mittelwert
	P_feed_L3_m	Float32		Wirkleistung Einspeisung L3 Mittelwert
	Q_feed_L1_m	Float32		Blindleistung Einspeisung L1 Mittelwert
	Q_feed_L2_m	Float32		Blindleistung Einspeisung L2 Mittelwert
	Q_feed_L3_m	Float32		Blindleistung Einspeisung L3 Mittelwert
	THD_U123_m	Float32		Gesamtverzerrung Spannung Mittelwert
	U_L1	Float32		Spannung L1—N Momentanwert
	U_L2	Float32		Spannung L2—N Momentanwert
	U_L3	Float32		Spannung L3—N Momentanwert
	I_feed_L1	Float32		Strom Einspeisung L1 Momentanwert
	I_feed_L2	Float32		Strom Einspeisung L2 Momentanwert

Kat.	PV-Name	Typ	Verfügb.	Erläuterung
	I_feed_L3	Float32		Strom Einspeisung L3 Momentanwert
	P_feed_L1	Float32		Wirkleistung Einspeisung L1 Momentanwert
	P_feed_L2	Float32		Wirkleistung Einspeisung L2 Momentanwert
	P_feed_L3	Float32		Wirkleistung Einspeisung L3 Momentanwert
	S_feed_L1	Float32		Scheinleistung Einspeisung L1 Momentanwert
	S_feed_L2	Float32		Scheinleistung Einspeisung L2 Momentanwert
	S_feed_L3	Float32		Scheinleistung Einspeisung L3 Momentanwert
	THD_U_L1	Float32		Gesamtverzerrung Spannung L1 Momentanwert
	THD_U_L2	Float32		Gesamtverzerrung Spannung L2 Momentanwert
	THD_U_L3	Float32		Gesamtverzerrung Spannung L3 Momentanwert
	THD_I_feed_L1	Float32		Gesamtverzerrung Strom Einspeisung L1 Mom.
	THD_I_feed_L2	Float32		Gesamtverzerrung Strom Einspeisung L2 Mom.
	THD_I_feed_L3	Float32		Gesamtverzerrung Strom Einspeisung L3 Mom.
	U_L1_min	Float32		Spannung L1—N Minimum
	U_L2_min	Float32		Spannung L2—N Minimum
	U_L3_min	Float32		Spannung L3—N Minimum
	I_feed_L1_min	Float32		Strom Einspeisung L1 Minimum
	I_feed_L2_min	Float32		Strom Einspeisung L2 Minimum
	I_feed_L3_min	Float32		Strom Einspeisung L3 Minimum
	P_feed_L1_min	Float32		Wirkleistung Einspeisung L1 Minimum
	P_feed_L2_min	Float32		Wirkleistung Einspeisung L2 Minimum
	P_feed_L3_min	Float32		Wirkleistung Einspeisung L3 Minimum
	Q_feed_L1_min	Float32		Scheinleistung Einspeisung L1 Minimum
	Q_feed_L2_min	Float32		Scheinleistung Einspeisung L2 Minimum
	Q_feed_L3_min	Float32		Scheinleistung Einspeisung L3 Minimum
	THD_U123_min	Float32		Gesamtverzerrung Spannung L1 Minimum
	U_L1_max	Float32		Spannung L1—N Maximum
	U_L2_max	Float32		Spannung L2—N Maximum
	U_L3_max	Float32		Spannung L3—N Maximum
	I_feed_L1_max	Float32		Strom Einspeisung L1 Maximum
	I_feed_L2_max	Float32		Strom Einspeisung L2 Maximum
	I_feed_L3_max	Float32		Strom Einspeisung L3 Maximum

Kat.	PV-Name	Typ	Verfügb.	Erläuterung
	P_feed_L1_max	Float32		Wirkleistung Einspeisung L1 Maximum
	P_feed_L2_max	Float32		Wirkleistung Einspeisung L2 Maximum
	P_feed_L3_max	Float32		Wirkleistung Einspeisung L3 Maximum
	Q_feed_L1_max	Float32		Scheinleistung Einspeisung L1 Maximum
	Q_feed_L2_max	Float32		Scheinleistung Einspeisung L2 Maximum
	Q_feed_L3_max	Float32		Scheinleistung Einspeisung L3 Maximum
	THD_U123_max	Float32		Gesamtverzerrung Spannung Maximum
Betriebsstundenzähler-Option			*grid4	s. Abschnitt 5.9
	unit1_op_time	32Bit		Laufzeit von Betriebsmittel 1 [s]
	unit2_op_time	32Bit		Laufzeit von Betriebsmittel 2 [s]
	single_op_time	32Bit		Laufzeit mit einem einzelnen Betriebsmittel [s]
	double_op_time	32Bit		Laufzeit mit beiden Betriebsmitteln [s]
	unit1_startcnt	32Bit		Startzähler Betriebsmittel 1 [s]
	unit2_startcnt	32Bit		Startzähler Betriebsmittel 2 [s]
	single_startcnt	32Bit		Startzähler Zyklen mit 1 Betriebsmittel [s]
	double_startcnt	32Bit		Startzähler Zyklen mit beiden Betriebmitteln [s]
	unit1_I_avg	Float32		Betriebsmittel 1 mittlere Auslastung
	unit1_I_min	Float32		Betriebsmittel 1 minimale Auslastung
	unit1_I_max	Float32		Betriebsmittel 1 maximale Auslastung
	unit2_I_avg	Float32		Betriebsmittel 2 mittlere Auslastung
	unit2_I_min	Float32		Betriebsmittel 2 minimale Auslastung
	unit2_I_max	Float32		Betriebsmittel 2 maximale Auslastung

5.13 Bitfelder

Einige der in der Firmware verwendeten Statusinformationen bestehen nur aus einem Bit oder einigen wenigen Bits. Diese Variablen werden für den Versand als LoRaWAN-Telegramm innerhalb der izi-io zu Bitfeldern zusammengepackt, die als Integer-Prozessvariablen mit einem oder mehreren Nibbles[4] Länge verarbeitet und so auch ins Sendetelegramm eingefügt werden. Je nach Firmwareversion können bestimmte Variablen mit Bitfeldern bereits vordefiniert sein, etwa `IOStatus`, `SysStatus_Tel0` und `SysStatus_Tel123`. Damit der Payload Decoder die Variablen wieder in Bitfelder zer-

4 Nibble: Halb-Byte à 4 Bits

legen und diese den entsprechenden Prozessvariablen zuweisen kann, sind diese Definitionen innerhalb der JSON-Datei anzugeben.

Unter dem Key **`Bit_Items`** können die Sub-Byte-Layouts für die Bitfeld-Prozessvariablen angegeben werden. Es handelt sich um eine Liste, die pro Eintrag eine Struktur mit folgenden Keys enthält:

`PV_name` [String]
Name der Variablen mit dem Bitfeld, das zerlegt werden soll.

`bitlength` [Integer]
Länge der Variablen in Bits, muss ein Vielfaches von 4 sein.

`fields` Eine Liste der in der Variablen enthaltenen Bits / Bitfelder, beginnend mit Bit 0 – siehe folgende Aufstellung.

Jeder Listeneintrag in **`fields`** stellt ein Bitfeld dar und hat folgende Keys:

`bitlength` [Integer]
Länge des Bitfeldes in Bits

`Alias` [String]
Name des Feldes

`Errormode` [String]
(optional:) Wenn vorhanden, wird diesem Bitfeld die Bezeichnung eines Fehlermodus zugeordnet. Das bedeutet, dieses Bitfeld codiert einen Fehlermodus, der den Gültigkeits- bzw. Fehlerstatus für andere Prozessvariablen anzeigt. In den Telegrammfeldern (Abschnitt 5.15) kann darauf Bezug genommen werden.

`Description` [String]
Beschreibung; dient ausschließlich der Dokumentation.

Falls Bits übrig bleiben, sind diese undefiniert und werden vom Payload Decoder nicht ausgewertet. Es dürfen maximal so viele Bits zugeordnet werden wie in der Variablen enthalten sind.

Als **`Errormode`** ist aktuell lediglich **`Def_Modbus`** `*grid` zulässig. Dieser Wert gibt an, dass das damit gekennzeichnete 2-Bit-Feld (d.h. es muss hier `"bitlength": 2` angegeben sein) den Fehlerzustand der Modbus-Kopplung codiert. Außerdem werden die Flags

`modbus_e_error` (Energiefehler) und `modbus_pq_error` (Power-Quality-Fehler) vom Payload Decoder anhand dieses Codes belegt.

Das mit **`Def_Modbus`** bezeichnete Bitfeld kann folgende Werte annehmen:

Wert	Bedeutung	**`modbus_e_error`** (Energiefehler)	**`modbus_pq_error`** (Power-Quality-Fehler)
0	kein Fehler; Energiemessdaten und PQ-Daten ok	`false`	`false`
1	nur Energiemessdaten; PQ-Daten liegen nicht vor	`false`	`true`
2	keine Daten (keine Verbindung)	`true`	`true`
3	interner Fehler	`true`	`true`

Zu beachten ist, dass anhand der beiden Fehler-Flags nicht zwischen Fall 2 und Fall 3 unterschieden werden kann. Dies kann aber in der weiteren Auswertung (d.h. hinter dem Payload Decoder) geschehen, da der Wert des Bitfelds unter dem gewählten Aliasnamen in der JSON-Rückgabestruktur enthalten ist. Eine weitere Differenzierung ist darüber hinaus durch Auswertung der Fehlercodes möglich, die in anderen Datenfeldern codiert werden können (vgl. Abschnitt 5.15.2, **`Errormode`**-Werte mit Suffix **`_C`**).

Felder, die als Flag die Gültigkeit einzelner Prozessvariablen angeben, werden in der Beschreibung des Telegrammfelds, das die Prozessvariable codiert, angegeben. Sie werden in den **`Bit_Items`** nicht gesondert gekennzeichnet.

Siehe hierzu auch die Beschreibung des korrespondierenden Keys **`Errormode`** in den Datenfeldern, Abschnitt 5.15.2.

5.13.1 Definierte Bitfelder

Die Prozessvariablen, die als Bitfeld definiert sind, werden hier spezifiziert. Die Feldnamen sind anders als die PV-Namen *nicht* fest in der Firmware vorgegeben, sondern werden durch die JSON-Struktur unterhalb **`Bit_Items`** erst definiert und werden damit lediglich in der Ausgabestruktur des Payload Decoders sichtbar. Die angegebenen Feldnamen sind mithin eine Empfehlung. Die dazugehörige JSON-Substruktur wird unter der Tabelle als Referenz ebenfalls angegeben.

`SysStatus_Tel0`

Systemstatus-Bitset, ausführliche Version

Länge: 8 Bit

Bit-Position	Bit-Länge	Feldname	Erläuterung
.0	2	DataStatus	ErrorMode Def_Modbus, vgl. 5.15.2
.2	1	newConfigLoaded	1 = neue Konfiguration wurde geladen
.3	2	StatusLoRaWAN	Status LoRaWAN-Funkmodul: 0 = running 1 = recovery from silent 2 = recovery from fatal error 3 = reset after command error
.5	2	ResetStatCPU	Ursache des letzten CPU-Resets: 0 = no reset 1 = power-on reset 2 = reset button pressed 3 = watchdog/software/security reset
.7	1	– unbenutzt –	—


```
{
    "PV_name":          "SysStatus_Tel0",
    "bitlength":        8,
    "fields":
    [
        {
            "bitlength":   2,
            "Alias":       "DataStatus",
            "Errormode":   "Def_Modbus",
            "Description":  "Energy / PQ data Status: 0= Energy and PQ data OK; 1= energy data only; 2 = no connection; 3 = internal errormode"
        },
        {
            "bitlength":   1,
            "Alias":       "newConfigLoaded",
            "Description":  "H = a new configuration file was loaded"
        },
        {
            "bitlength":   2,
            "Alias":       "StatusLoRaWAN",
            "Description":  "LoRaWAN radio module error Status: 0= running; 1= recovery from silent; 2= recovery from fatal error; 3= reset after command error"
        },
        {
            "bitlength":   2,
            "Alias":       "ResetStatCPU",
            "Description":  "Reset Status: 0= no Reset; 1= power on reset; 2= reset button; 3= watchdog /software / security reset"
        }
    ]
```

```
}
```

SysStatus_Tel123

Systemstatus-Bitset, knappe Version

Länge: 8 Bit

Bit-Position	Bit-Länge	Feldname	Erläuterung
.0	2	DataStatus	ErrorMode Def_Modbus, vgl. 5.15.2
.2	6	– unbenutzt –	—

```
{
    "PV_name":              "SysStatus_Tel123",
    "bitlength":            8,
    "fields":
    [
        {
            "bitlength":    2,
            "Alias":        "DataStatus",
            "Description":  "Energy / PQ data Status: 0= Energy and PQ data OK; 1= energy data only; 2 = no connection; 3 = internal errormode"
        }
    ]
}
```

IOStatus

IO-Status-Bitset

Länge: 8 Bit

Bit-Position	Bit-Länge	Feldname	Erläuterung
.0	1	AI0_stat / DI0	– siehe unten –
.1	1	AI1_stat / DI1	
.2	1	AI2_stat / DI2	
.3	1	AI3_stat / DI3	
.4	1	AI4_stat / DI4	
.5	1	AI5_stat / DI5	
.6	1	AI6_stat / DI6	
.7	1	AI7_stat / DI7	

Die Bedeutung des `IOStatus`-Bits eines Eingangs unterscheidet sich abhängig davon, wie dieser Eingang konfiguriert ist. Für Eingänge, die als Analog-Eingang konfiguriert sind, enthält das Bit den Fehler-Status des Analogwerts (0= Messwert gültig; 1 = Fehler). Für Digitaleingänge enthält das Bit den Eingangswert (H-Pegel oder L-Pegel). Die angegebenen Feldnamen sollten daher passend zur gewählten Eingangskonfiguration vergeben werden.

```
{
    "PV_name":              "IOStatus",
    "bitlength":            8,
    "fields":
    [
        {
            "bitlength":    1,
            "Alias":        "AI0_stat",
            "Description":  "Fehlerflag Analogeingang Kanal 0"
        },
        {
            "bitlength":    1,
            "Alias":        "DI1",
            "Description":  "Digitaleingang Kanal 1"
        },
        {
            "bitlength":    1,
            "Alias":        "DI2",
            "Description":  "Digitaleingang Kanal 2"
        },
        {
            "bitlength":    1,
            "Alias":        "DI3",
            "Description":  "Digitaleingang Kanal 3"
        },
        {
            "bitlength":    1,
            "Alias":        "DI4",
            "Description":  "Digitaleingang Kanal 4"
        },
        {
            "bitlength":    1,
            "Alias":        "DI5",
            "Description":  "Digitaleingang Kanal 5"
        },
        {
            "bitlength":    1,
            "Alias":        "DI6",
            "Description":  "Digitaleingang Kanal 6"
        },
        {
            "bitlength":    1,
```

```
43              "Alias":        "DI7",
44              "Description":  "Digitaleingang Kanal 7"
45          }
46      ]
47  }
```

IO_Loopcheck_Stat *grid6

Loopcheck-Status-Bitset

Länge: 8 Bit

Bit-Position	Bit-Länge	Feldname	Erläuterung
.0	1	DI0_chk_OK	Loopcheck IOØ bestanden
.1	1	DI1_chk_OK	Loopcheck IO1 bestanden
.2	1	DI2_chk_OK	Loopcheck IO2 bestanden
.3	1	DI3_chk_OK	Loopcheck IO3 bestanden
.4	1	DI4_chk_OK	Loopcheck IO4 bestanden
.5	1	DI5_chk_OK	Loopcheck IO5 bestanden
.6	1	DI6_chk_OK	Loopcheck IO6 bestanden
.7	1	DI7_chk_OK	Loopcheck IO7 bestanden
.8	1	Loopchk_timeout	Loopcheck: Timeout-Fehler
.9	1	Loopchk_short	Loopcheck: Querschluss-Fehler
.10	1	Loopchk_pass	Loopcheck bestanden
.11 ... .15	5	– unbenutzt –	

Beispiel: Nach einem erfolgreichen Prüfablauf hat das Status-Bitset für den Loopcheck von IO0, IO1, IO2, IO3 den Inhalt `0b0000010000001111` = `0x040F` = 1039.

```
1   {
2       "PV_name":              "IO_Loopcheck_Stat",
3       "bitlength":            8,
4       "fields":
5       [
6           {
7               "bitlength":    1,
8               "Alias":        "DI0_chk_OK",
9               "Description":  "Loopcheck IO0 bestanden"
10          }, {
11              "bitlength":    1,
12              "Alias":        "DI1_chk_OK",
13              "Description":  "Loopcheck IO1 bestanden"
```

```
14          }, {
15              "bitlength":    1,
16              "Alias":        "DI2_chk_OK",
17              "Description":  "Loopcheck IO2 bestanden"
18          }, {
19              "bitlength":    1,
20              "Alias":        "DI3_chk_OK",
21              "Description":  "Loopcheck IO3 bestanden"
22          }, {
23              "bitlength":    1,
24              "Alias":        "DI4_chk_OK",
25              "Description":  "Loopcheck IO4 bestanden"
26          }, {
27              "bitlength":    1,
28              "Alias":        "DI5_chk_OK",
29              "Description":  "Loopcheck IO5 bestanden"
30          }, {
31              "bitlength":    1,
32              "Alias":        "DI6_chk_OK",
33              "Description":  "Loopcheck IO6 bestanden"
34          }, {
35              "bitlength":    1,
36              "Alias":        "DI7_chk_OK",
37              "Description":  "Loopcheck IO7 bestanden"
38           }, {
39              "bitlength":    1,
40              "Alias":        "Loopchk_timeout",
41              "Description":  "Loopcheck Fehler timeout"
42          }, {
43              "bitlength":    1,
44              "Alias":        "Loopchk_short",
45              "Description":  "Loopcheck Fehler Querschluss"
46          }, {
47              "bitlength":    1,
48              "Alias":        "Loopchk_pass",
49             "Description":   "Loopcheck bestanden"
50          }, {
51              "bitlength":    1,
52              "Alias":        "Loopchk_timeout",
53              "Description":  "Loopcheck: Timeout-Fehler"
54          }, {
55              "bitlength":    1,
56              "Alias":        "Loopchk_short",
57              "Description":  "Loopcheck: Querschluss-Fehler"
58          }, {
59              "bitlength":    1,
60              "Alias":        "Loopchk_pass",
61              "Description":  "Loopcheck bestanden"
62          }
```

```
63          ]
64  }
```

5.14 Triggerbedingungen und Sendeablauf

Jede der vordefinierten Prozessvariablen ist in der Firmware mit einer Triggerbedingung verbunden. Standardmäßig wird dieser Trigger bei einer Änderung der Prozessvariablen um eine parametrierbare Differenz gegenüber dem im letzten Telegramm ausgesandten Wert ausgelöst.

Neben diesen Standard-Triggern können einige Prozessvariablen davon abweichend andere Triggerbedingungen haben. Ein Beispiel sind die Funktionen zur Überwachung der Messwerte eines elektrischen Netzes, wie sie in Abschnitt 5.7 beschrieben werden.

Ein Telegramm wird immer dann in die Warteschlange zur Aussendung eingereiht,

- wenn die globale Inhibit-Zeit (siehe unten) abgelaufen ist und
- wenn die Inhibit- Zeit des Telegramms abgelaufen ist und
- wenn der Trigger bei einer der in diesem Telegramm vorkommenden Prozessvariablen gesetzt ist.

Mit der Aussendung eines Telegramms starten zwei Inhibit-Zeiten:

- eine globale Inhibit-Zeit: Während diese Sperrzeit läuft, können gar keine Telegramme, egal welchen Telegrammtyps, ausgesendet werden; siehe Abschnitt 5.4.
- eine für den Telegrammtyp spezifische Inhibit-Zeit: Solange diese Sperrzeit läuft, können keine Telegramme dieses Typs ausgesendet werden; siehe Abschnitt 5.15.1.

Für jeden Telegrammtyp getrennt wird maximal ein noch nicht gesendetes Telegramm zwischengespeichert. Wenn dieser Speicher eines Telegrammtyps bereits besetzt ist und für diesen Telegrammtyp noch eine weitere Sendeanforderung eingeht, werden die älteren Daten jeweils durch die neuen Daten verdrängt.

Wenn für die Telegramm-spezifische Zeitdauer **`Event_time`** kein Triggerereignis eingetreten ist, wird dennoch eine Aussendung ausgelöst. Auf diese Weise können auch bei selten auftretenden Ereignissen regelmäßig LoRaWAN-Telegramme erzeugt werden und die Funktion der Übertragungsstrecke kann anhand der regelmäßig eingehenden Telegramme serverseitig überprüft werden.

5.15 Definition der LoRaWAN-Telegramme

Der Aufbau der LoRaWAN-Telegramme ist über die JSON-Datei frei konfigurierbar. Aus der Liste der in der izi-io vorhandenen Prozessvariablen kann für jeden Telegrammtyp eine Zusammenstellung von Variablen ausgewählt werden. Konfigurierbar sind ferner der Wertebereich und die Bitauflösung, mit der jede Prozessvariable über LoRaWAN übertragen wird.

Unter dem Key **`LoRaWAN_Telegrams`** werden in einer Liste die möglichen Telegrammtypen definiert. Jeder Listeneintrag ist einem Telegrammtyp zugeordnet. In der aktuellen Firmwareversion können maximal 4 verschiedene Telegrammtypen definiert werden.

5.15.1 Allgemeine Angaben zu einem Telegrammtyp

Folgende Key-Value-Paare werden zu jedem Telegrammtyp auf der obersten Ebene definiert:

`TelegramType` [String]
Typ des Telegramms (Kurzbezeichnung – dient ausschließlich der Dokumentation).

`Description` [String]
Beschreibung; dient ausschließlich der Dokumentation.

`Priority` [Flag]
Flag, ob es sich um ein priorisiertes Telegramm handelt.
- **`true`**: Das Telegramm wird als *confirmed* versendet oder es wird wiederholt, siehe hierzu auch Abschnitt 5.4, Parameter **`PrioMode`**.
- **`false`**: Das Telegramm wird unbestätigt einmal ausgesendet.

`PrioOnEventsOnly` [Flag]
Gibt an, ob als priorisiert gekennzeichnete Telegramme (s.o. **`Priority`**) nur dann mit Priorität gesendet werden, wenn sie aufgrund eines Events ausgelöst wurden.
- **`true`**: Zyklische Aussendungen werden ohne Priorität gesendet.
- **`false`**: Auch zyklische Aussendungen werden mit Priorität gesendet.

LoRaWAN_PortNo [Integer]
LoRaWAN-Portnummer des Telegramms.

Event_time [Integer]
Maximaler Zeitabstand in Sekunden zwischen zwei Aussendungen des Telegramms, siehe Abschnitt 5.14. Mit **Event_time** = 0 wird diese Funktion deaktiviert.

Inhibit_time [Integer]
Sperrzeit für die Aussendung des Telegramms in Sekunden. Um zu verhindern, dass bei häufig auftretenden Triggerereignissen das dazugehörige Telegramm zu oft ausgesendet wird, kann dieser Telegrammtyp erst nach Ablauf von **Inhibit_time** erneut ausgesendet werden, siehe Abschnitt 5.14.

5.15.2 Datenfelder

Unter dem Key **Data_Items** werden in einer Liste die einzelnen Datenfelder des Telegramms aufgelistet. Zu jedem Eintrag gibt es folgende Keys:

Alias [String]
(Optional:) Name der Prozessvariablen, der vom Payload Decoder an Stelle von **PV_name** in die JSON-Rückgabe geschrieben wird. Dies muss ein gültiger JavaScript-Bezeichner sein. Falls nicht angegeben, wird der **PV_name** verwendet.

Description [String]
Beschreibung; dient ausschließlich der Dokumentation.

PV_name [String]
Name der Prozessvariablen, wie er in der izi-io-Firmware vordefiniert ist. Die verfügbaren Prozessvariablen sind abhängig von der Firmwarevariante bzw. -version und sind in Abschnitt 5.12 aufgelistet.

PV_at_data_zero [Float]
Wert der Prozessvariablen, dem der Wert 0 des zugehörigen Telegramm-Datenfeldes zugeordnet ist; siehe unten.

PV_at_data_max [Float]
Wert der Prozessvariablen, dem der größte darstellbare Wert

des zugehörigen Telegramm-Datenfeldes zugeordnet ist; siehe unten. Bei einem 8-Bit-Datenfeld wäre der größte darstellbare Wert beispielsweise 255, bei 12 Bit 4.095, bei 16 Bit 65.535 und bei 32 Bit 4.294.967.295.

Valid_min [Float]
Die untere Grenze des gültigen Intervalls von Werten der Prozessvariablen; siehe unten. Bei Unterschreitung dieser Grenze wird die Prozessvariable in der Rückgabestruktur des Payload Decoders als ungültig erklärt.

Valid_max [Float]
Die obere Grenze des gültigen Intervalls von Werten der Prozessvariablen; siehe unten. Bei Überschreitung dieser Grenze wird die Prozessvariable als ungültig erklärt.

Trigger_delta [Float]
Die Bedeutung ist abhängig vom Typ der Prozessvariablen:
Float-Prozessvariable: Die Differenz, um die sich die Prozessvariable gegenüber dem zuletzt in einem Telegramm ausgesendeten Wert ändern muss, um ein asynchrones Meldetelegramm zu veranlassen. Wenn dieser Wert auf 0 gesetzt ist, ist die Triggerfunktion deaktiviert.
Integer-Prozessvariable:
0: Triggerfunktion deaktiviert.
1: Trigger, wenn ein Bit in der PV von 0 nach 1 wechselt.
2: Trigger, wenn ein Bit in der PV von 1 nach 0 wechselt.
3: Trigger, wenn ein Bit in der PV beliebig wechselt.
4: Trigger, wenn der Integer-Wert größer wird
5: Trigger, wenn der Integer-Wert kleiner wird.

bitlength [String]
Länge des Datenfeldes und dessen Codierung im LoRaWAN Telegramm. Die Prozessvariable kann als 4-Byte-Float oder in einem Integer-Format wählbarer Länge (4, 8, 12, 16 oder 32 Bit) übertragen werden. In letzterem Fall wird der zwischen **PV_at_data_zero** und **PV_at_data_max** liegende Wertebereich der Prozessvariablen auf die Integervariable im Telegramm abgebildet. Mögliche Parameter für **bitlength** sind:

- **4Bit**
- **8Bit**
- **12Bit**
- **16Bit**
- **32Bit**
- **Float32**
- **disabled**

Unit [String]
Physikalische Maßeinheit der Prozessvariablen. Diese Angabe wird vom Payload Decoder in die Ausgabestruktur des Datenfelds übernommen.

Errormode [String]
Gibt an, ob das Datenfeld einer speziellen Fehlerbehandlung unterzogen wird. Mögliche Werte:

- **bit** *<bsi>* Wenn das Flag *<bsi>* gesetzt ist (Wert = 1), ist das Datenfeld ungültig. *<bsi>* bezeichnet ein Bitset-Item, ist also ein **Alias** in einem **Bit_Item** – vgl. Abschnitt 5.12
- **Modbus_E** `*grid` Das mit **Def_Modbus** gekennzeichnete Feld in den **Bit_Items** enthält den Modbus-Fehlercode. Wenn ein Fehler der Energie-Messdaten vorliegt, ist dieses Datenfeld ungültig.
- **Modbus_E_C** `*grid` Das mit **Def_Modbus** gekennzeichnete Feld in den **Bit_Items** enthält den Modbus-Fehlercode. Wenn ein Fehler der Energie-Messdaten vorliegt, ist dieses Datenfeld ungültig und enthält stattdessen den Fehlercode.
- **Modbus_PQ** `*grid` Das mit **Def_Modbus** gekennzeichnete Feld in den **Bit_Items** enthält den Modbus-Fehlercode. Wenn ein Fehler der PowerQuality-Daten vorliegt oder diese nicht zur Verfügung stehen, ist dieses Datenfeld ungültig.
- **Modbus_PQ_C** `*grid` Das mit **Def_Modbus** gekennzeichnete Feld in den **Bit_Items** enthält den Modbus-Fehlercode. Wenn ein Fehler der PowerQuality-Daten vorliegt oder diese nicht zur Verfügung stehen, ist dieses Datenfeld ungültig und enthält stattdessen den Fehlercode.

Referenzierte *<bsi>*-Variablen müssen im selben Telegramm vorhanden und eindeutig sein, d.h. genau ein Bitset, in dem sie lt. **`Bit_Item`**-Definition als **`PV_name`** enthalten sind, muss im Telegramm vorkommen.

Die mit **`Modbus_E/PQ_C`** gekennzeichneten Datenfelder transportieren im Nicht-Fehlerfall einen Messwert, im Fehlerfall hingegen einen Fehlercode. Der jeweilige numerische Fehlercode (d.h. **E** respektive **PQ**) wird über ein zusätzliches JSON-Feld `modbus_e_errorcode` resp. `modbus_pq_errorcode` ausgegeben. Im Nicht-Fehlerfall ist der übermittelte Fehlercode `0`.

Zur Feldlänge (**`bitlength`**): Feldgrößen sind stets ganzzahlige Vielfache von Nibbles, d.h 4-Bit-Abschnitten. Mögliche Feldgrößen sind 4, 8, 12, 16 und 32 Bit. Außerdem sind Floats im 32-Bit-IEEE-Format (single precision) zulässig. Als Encoding wird *Little Endian* verwendet, d.h. LSB first.

Es ergibt sich folgendes Schema:

Datentyp	**Telegrammbytes (hex)** `Index [0] [1] [2] [3]`	**Ergebnis (hex)**
12 Bit ab Bit 0	`12 34`	`412`
12 Bit ab Bit 4	`12 34`	`341`
16 Bit	`12 34`	`3412`
24 Bit	`12 34 56`	`563412`
32 Bit	`12 34 56 78`	`78563412`

Zu **`PV_at_data_zero`**/**`max`**: Hiermit kann von allen Werten, die eine Prozessvariable in der Firmware haben kann, ein sinnvoller Ausschnitt angegeben werden, der linear in das Telegrammfeld mit seinem Roh-Wertebereich (0 bis 2^{BitLen} -1) skaliert wird.

Beispiel:
Die Prozessvariable ist in der izi-io intern ein Float-Wert, der eine Temperatur in °C enthält. Wenn für die Anwendung etwa nur Werte von -50°C bis 100°C von Belang sind, wird auch nur dieser Wertebereich in ein 16-Bit-Feld codiert. Der 16-Bit-Wert 0 entspricht in diesem Fall einer Temperatur von -50 °C und der Wert 65535 einer Temperatur von 100 °C. Hierzu wäre dann der Parameter **`PV_at_data_zero`** auf -50 und **`PV_at_data_max`** auf 100 zu setzen. Der Payload Decoder rechnet dann automatisch wieder um in die Temperatur.

Mit **`Valid_min/max`** kann optional der zulässige Bereich valider Messwerte eingegrenzt werden. Wenn diese Parameter angegeben werden, prüft der Payload Decoder, ob der übermittelte Wert außerhalb des gültigen Bereichs liegt, und setzt im JSON dementsprechend das `error`-Flag für den Messwert.

Beispiel:
Für die im vorherigen Beispiel genannte Temperaturmessung sollen ganze Gradzahlen übermittelt werden. In diesem Fall ist ein 8-Bit-Feld ausreichend, und man kann **`PV_at_data_zero`** auf -50 und **`PV_at_data_max`** auf 205 setzen (die Differenz ist genau gleich 255), um die gewünschte Quantisierung zu erreichen. Zusätzlich wird nun **`Valid_max`** auf 100 gesetzt, um die im Feld codierbaren Werte von 101 bis 255 als ungültig zu kennzeichnen. **`Valid_min`** wird hier nicht benötigt, da es am unteren Ende des Wertebereichs keine ungültigen Werte gibt (0 entspricht den gewünschten -50°C).

5.15.3 Berechnete Felder

Unter dem Key **`Calc_Items`** können aus den im jeweiligen Telegramm übermittelten Datenfeldern (inklusive der Unterfelder bei Bitsets sowie der zuvor errechneten Werte) weitere Werte errechnet werden. Dies geschieht nach dem Empfang im Payload Decoder. Es wird eine Liste angegeben, die jeweils eine Struktur mit folgenden Keys enthält:

`Name` [String]
Name der berechneten Größe. Dies muss ein gültiger JavaScript-Identifier sein, d.h. als erstes Zeichen ist ein Buchstabe (a..z, A..Z)[5], ein Unterstrich (_) oder das Dollarzeichen ($) zulässig, für die weiteren Zeichen zusätzlich auch Ziffern (`0`..`9`). Bezeichner sind case-sensitive, d.h. Groß-/Kleinschreibung ist relevant.

`Expr` [String]
JavaScript-Ausdruck zur Berechnung. Es stehen die JavaScript-üblichen Funktionen zur Verfügung, ggf. mit Präfix „`Math.`“, also etwa „`Math.sqrt(…)`“ für die Quadratwurzelfunktion. Die Berechnung muss funktional als einzelner Ausdruck erfolgen, d.h. es können nicht mehrere Statements angegeben werden.

5 Theoretisch sind auch Non-ASCII-Zeichen, also erweitertes ASCII wie Latin-1 oder Unicode zulässig, dies wird aber nicht empfohlen. Maßgeblich ist der verwendete JavaScript-Dialekt, im Falle des The Things Stack V3 ist dies aktuell ECMAScript 2009 (ES5); außerdem könnten je nach Umgebung weitere Einschränkungen bestehen.

Unit [String]
Physikalische Maßeinheit der Größe.

Bei der Generierung des Payload Decoders werden die JavaScript-Ausdrücke nicht auf syntaktische Korrektheit überprüft, sondern sie werden lediglich in den entsprechenden Abschnitt des Scripts übernommen. Dabei erfolgt eine textuelle Ersetzung der in den Ausdrücken verwendeten Variablen durch deren Darstellung in der Decoder-Funktion, d.h. aus der Variablen `varname` wird dann etwa `d["varname"].value`. Zusätzlich wird Code erzeugt, der den Fehlerstatus der verwendeten Variablen ermittelt, abhängig davon den Ausdruck berechnet sowie nach der Berechnung prüft, ob ein numerisch valides Ergebnis erhalten wurde. Abhängig davon wird dann das `error`-Flag des berechneten Wertes gesetzt.

Beispiel:

Im Telegramm werden die Felder `Wirkleistung` und `Scheinleistung` übermittelt. Als **`Calc_Items`** wurde deklariert:

```
"Calc_Items": [
  {
    "Name": "Blindleistung",
    "Expr": "Math.sqrt(Math.pow(Scheinleistung,2) - Math.pow(Wirkleistung,2))",
    "Unit": "VA"
  },
  {
    "Name": "Leistungsfaktor",
    "Expr": "Math.abs(Wirkleistung) / Scheinleistung"
  },
]
```

Daraus wird folgender Code erzeugt:

```
if (port == …) {
  var val, err;
  if (d["Wirkleistung"].error || d["Scheinleistung"].error) {
    d["Blindleistung"] = {value:null, unit:"VA", error:true};
  } else {
    val = Math.sqrt(Math.pow(d["Scheinleistung"].value,2)
          - Math.pow(d["Wirkleistung"].value,2));
    err = !ok(val);
    d["Blindleistung"] = {value:(err?null:val), unit:"VA", error:err};
  }
  if (d["Wirkleistung"].error || d["Scheinleistung"].error) {
    d["Leistungsfaktor"] = {value:null, unit:"", error:true};
  } else {
    val = Math.abs(d["Wirkleistung"].value) / d["Scheinleistung"].value;
    err = !ok(val);
    d["Leistungsfaktor"] = {value:(err?null:val), unit:"", error:err};
  }
}
```

Achtung: Die Berechnung mit den notwendigen Fehlerprüfungen ist recht umfangreich und führt zu einem entsprechenden Umfang des Payload Decoders. Dies ist zu berücksichtigen, wenn es beim genutzten LoRaWAN-Netzwerkserver Längenbegrenzungen für den Payload Decoder gibt.

6 Payload Formatter / Decoder

6.1 Mechanismus

Das Konfigurationstool erzeugt aus der Eingabedatei eine JavaScript-Datenstruktur namens `conf`, mit der die darauf folgenden Funktionen des Templates parametriert werden. Zusätzlich wird anhand der Aufrufoptionen des Tools das Flag `cmp` gesetzt, das festlegt, ob die ausgegebene JSON-Struktur *kompakt* sein soll (siehe unten).

Die berechneten Werte werden als generierter Code an der mit `//#Calc_Items#//` bezeichneten Stelle eingefügt.

Der verwendete JavaScript-Dialekt ist ECMAScript 2009 (ES5).

Das Template für den Payload Formatter sieht wie folgt aus (zu füllende Abschnitte farbig gekennzeichnet):

```
conf = { ..... };
cmp = ..... ;
// ~~~ no changes needed below this line ~~~
function ok(x) { return x!==null && isFinite(x) && !isNaN(x); }
function Decoder(bytes, port) {
  bytes.reverse();
  var istr=Array.from(bytes,
    function(b) {return('0'+(b&0xFF).toString(16)).slice(-2);}).join('');
  var d={};
  for (let i in conf[port]) {
    var it=conf[port][i];
    var nm=it[0],po=it[1],ln=it[2],un=it[3],a=it[4],
          b=it[5],vmin=it[6],vmax=it[7],bits=it[8],ecnf=it[9];
    v=po?istr.slice(-po-ln,-po):istr.slice(-ln);
    v=parseInt(v,16);
    if (bits) {
      d[nm]=cmp?v:{value:v,unit:""};
      for (let j in bits) {
        var bi=bits[j];
        var nm=bi[0], po=bi[1], ln=bi[2],ec=bi[3];
        bv=(v>>po)&((1<<ln)-1);
        d[nm]=cmp?bv:{value:bv,unit:""};
        if (ec=="def_m") {
          d["modbus_e_error"]=(bv&3)>1;
          d["modbus_pq_error"]=(bv&3)>0;
        }
      }
    } else {
      vv=+((a*v+b).toFixed(2));
      cerr = 0;
      if (ecnf!==null) {
```

```
32          var emode=ecnf[0],eref=ecnf[1];
33          if (emode=="bit")
34            cerr=cmp?d[eref]:d[eref].value;
35          else if (emode=="m_e")
36            cerr=d["modbus_e_error"];
37          else if (emode=="m_ec") {
38            cerr=d["modbus_e_error"];
39            d["modbus_e_errorcode"]=cerr?v:0;
40          } else if (emode=="m_pq")
41            cerr=d["modbus_pq_error"];
42          else if (emode=="m_pqc") {
43            cerr=d["modbus_pq_error"];
44            d["modbus_pq_errorcode"]=cerr?v:0;
45          } else
46            cerr=0;
47        }
48        verr=(vmin!==null&&vmax!==null)&&((vv<vmin)||(vv>vmax));
49        err=cerr|verr;
50        d[nm]=cmp?(err?null:vv):{value:(err?null:vv),unit:un,error:err};
51      }
52    }
53    //#Calc_Items#//
54    return d;
55  }
56  function decodeUplink(input) {
57    return {data:Decoder(input.bytes, input.fPort),warnings:[],errors:[]};
58  }
```

Dabei ist die erste Funktionsdefinition (`function Decoder(bytes, port)`) für den Payload Decoder der inzwischen obsoleten TTN Console V2 vorgesehen. Der Wrapper darunter (`function decodeUplink(input)`) ist für den Einsatz mit The Things Stack V3 vorgesehen und setzt das dort verwendete API auf die „alte“ Version um. In einer späteren Version des Tools wird die V2-Unterstützung entfallen, dann wird direkt der V3-kompatible Code erzeugt.

Der Payload Decoder erhält vom aufrufenden System als Parameter ein Byte-Array der Telegrammdaten sowie die Portnummer, mit der das Telegramm gesendet wurde. Weitere Metadaten werden vom System nicht zur Verfügung gestellt.

Der Rückgabewert ist im Regelfall (d.h. nicht-kompaktes Format) eine JSON-Struktur (Dictionary), die für jedes extrahierte Datenfeld wiederum ein Dictionary enthält, bestehend aus den Komponenten `value` (dem numerischen Wert des Datenfeldes, falls gültig, ansonsten `null`), optional `error` (einem Boolean-Wert, der die Ungültigkeit angibt) sowie `unit` (einem String, der die Maßeinheit des Werts angibt). Zusätzlich können noch weitere Statuswerte übermittelt werden.

Alternativ kann, wenn das Flag `cmp` gesetzt ist, eine kompakte Variante der Datenstruktur erzeugt werden, die für jedes extrahierte Datenfeld direkt den numerischen Wert oder im Fehlerfall `null` enthält. Die Einstellung, ob `cmp` auf `true` oder `false` gesetzt wird, erfolgt über den Aufruf des Konfigurationstools. Unabhängig davon kann der Wert aber auch nachträglich modifiziert werden.

6.2 Beispiel

Beispiel für einen Decoder-Aufruf mit einem Byte-Array des Telegramms sowie der Portnummer:

```
Decoder([1, 230, 25, 0, 0, 240, 200, 220, 230, 240, 4, 255, 246, 243], 20);
```

→ zurückgegebene JSON-Datenstruktur:

```
{
  modbus_pq_error: true,
  modbus_e_error: false,
  DataStatus: { unit: '', value: 1 },
  Sys_LoRaWAN_rejoin: { unit: '', value: 0 },
  Sys_LoRaWAN_telgrErr: { unit: '', value: 0 },
  Sys_ResetStat: { unit: '', value: 0 },
  AI0_stat: { unit: '', value: 0 },
  AI1_stat: { unit: '', value: 1 },
  AI2_stat: { unit: '', value: 1 },
  AI3_stat: { unit: '', value: 0 },
  DI0: { unit: '', value: 0 },
  DI1: { unit: '', value: 1 },
  DI2: { unit: '', value: 1 },
  DI3: { unit: '', value: 1 },
  AI0_Temp0: { unit: '°C', error: false, value: -17.5 },
  AI1_Temp1: { unit: '°C', error: true, value: null },
  AI2_currloop0: { unit: '%', error: true, value: null },
  AI3_currloop1: { unit: '%', error: false, value: 80.375 },
  U_L1_m: { unit: 'V', error: false, value: 220 },
  modbus_e_errorcode: 0,
  U_L2_m: { unit: 'V', error: false, value: 230 },
  U_L3_m: { unit: 'V', error: false, value: 240 },
  THD_U123_m: { unit: '%', error: true, value: null },
  modbus_pq_errorcode: 4,
  P_feed_m: { unit: 'kW', error: false, value: 26726.4 },
  S_feed_m: { unit: 'kVA', error: false, value: 1561.2 }
}
```

6.3 Portierung

Analog zum oben beschriebenen Wrapper, der den Aufruf der für den V2-Stack vorgesehenen Funktion für den V3-Stack anpasst, indem Aufrufparameter und Rückgabestruktur modifiziert werden, kann dies auch für andere LoRaWAN-Stacks erfolgen, die einen auf JavaScript beruhenden Plugin-Mechanismus nutzen. Für Systeme in anderen Programmiersprachen ist die Umsetzung natürlich komplexer. Nehmen Sie hierzu gerne Kontakt mit uns auf.

A Beispiel für eine Konfigurationsdatei

Gezeigt wird hier exemplarisch eine typische Konfiguration für zwei **izi-io /grid**, bei denen die ersten beiden IO-Anschlüsse genutzt werden, der erste für einen Pt1000-Temperaturfühler und der zweite für ein SPS-Schaltsignal.

```
{
   "all_Devices": {
      "Name":         "izi-io/grid2",
      "Description":  "izi-io ONS Überwachung mit PL-Multi Ver. 2",
      "Firmwaretype": "grid_plmulti_v2",
      "RequiredVersions": {
         "Hardware": "1.1.0",
         "Kernel":   "1.0.0",
         "Firmware": "1.0.0"
      },

      "LoRaWAN_Param": {
         "Mode":       "EU868",
         "Class":      "A",
         "PrioMode":   "confirmed",
         "JoinMode":   "OTAA",
         "OTAAreboot": "dontforce",
         "DR":         "SF10/125kHz",
         "DR_RX2":     "SF9/125kHz",
         "AdaptDR":    "enabled",
         "AutomReply": "enabled",
         "t_linkchk":  14400,
         "t_inhibit":  60,
         "t_wait_err": 60,
         "Err_retr":   50,
         "Tx_power":   1,
         "confRetrNr": 4,
         "rxdelay1":   1000,
         "rx2_freq":   869525000,
         "synchWord":  52
      },

      "PL_Multi_Param": {
         "Float32_Format":    "CDAB",
         "Modbus_parity":     "none",
         "Modbus_serialMode": "RS485",
         "Modbus_Baudrate":   115200,
         "Modbus_stopbits":   1,
         "PL_Multi_SlaveAdr": 10
      },

```

```
42        "ACgrid_Param": {
43           "U_Lim_L":       207,
44           "U_Lim_H":       253,
45           "U_Lim_sym":      10,
46           "I_Lim_L":         5,
47           "I_Lim_H":       500,
48           "I_Lim_sym":      30,
49           "P_Lim_L":         2,
50           "P_Lim_H":       100,
51           "P_Lim_sym":      20,
52           "THD_U_Lim_H":     5,
53           "THD_I_Lim_H":    30,
54           "mean_interval": 900
55        },
56
57        "Input_Config": [{
58              "chanNo": 0,
59              "RLine":  0,
60              "Mode":   "RTD_Pt1000",
61              "Unit":   "°C"
62           },
63           {
64              "chanNo": 1,
65              "Mode":   "DIN_24V_PLC"
66           },
67        ],
68
69        "Bit_Items": [{
70              "PV_name": "SysStatus_Tel0",
71              "bitlength": 8,
72              "fields": [{
73                    "bitlength": 2,
74                    "Alias": "DataStatus",
75                    "Errormode": "Def_Modbus",
76                    "Description": "Energy / PQ data Status: 0= Energy and PQ data OK; 1=
  energy data only; 2 = no connection; 3 = internal errormode"
77                 },
78                 {
79                    "bitlength": 1,
80                    "Alias": "newConfigLoaded",
81                    "Description": "H = a new configuration file was loaded"
82                 },
83                 {
84                    "bitlength": 2,
85                    "Alias": "StatusLoRaWAN",
86                    "Description": "LoRaWAN radio module error Status: 0= running; 1= re-
  covery from silent; 2= recovery from fatal error; 3= reset after command error"
87                 },
88                 {
```

```
                    "bitlength": 2,
                    "Alias": "ResetStatCPU",
                    "Description": "Reset Status: 0= no Reset; 1= power on reset; 2= reset button; 3= watchdog /software / security reset"
                }
             ]
          },
          {
             "PV_name": "SysStatus_Tel123",
             "bitlength": 8,
             "fields": [{
                    "bitlength": 2,
                    "Alias": "DataStatus",
                    "Description": "Energy / PQ data Status: 0= Energy and PQ data OK; 1= energy data only; 2 = no connection; 3 = internal errormode"
                }
             ]
          },
          {
             "PV_name": "IOStatus",
             "bitlength": 8,
             "fields": [{
                    "bitlength": 1,
                    "Alias": "AI0_stat",
                    "Description": "Fehlerflag Analogeingang 0"
                },
                {
                   "bitlength": 1,
                   "Alias": "DI0",
                   "Description": "Digitaleingang 0"
                },
                {
                   "bitlength": 1,
                   "Alias": "DI1",
                   "Description": "Digitaleingang 1"
                },
                {
                   "bitlength": 1,
                   "Alias": "DI2",
                   "Description": "Digitaleingang 2"
                },
                {
                   "bitlength": 1,
                   "Alias": "DI3",
                   "Description": "Digitaleingang 3"
                },
                {
                   "bitlength": 1,
                   "Alias": "DI4",
```

```
                  "Description": "Digitaleingang 4"
               },
               {
                  "bitlength": 1,
                  "Alias": "DI5",
                  "Description": "Digitaleingang 5"
               },
               {
                  "bitlength": 1,
                  "Alias": "DI6",
                  "Description": "Digitaleingang 6"
               }
            ]
         }
      ],

      "LoRaWAN_Telegrams": [{
            "TelegramType": "Standardmeldung",
            "Description": "zyklisches Messwerttelegramm mit Mittelwerten",
            "Priority": false,
            "LoRaWAN_PortNo": 20,
            "Event_time": 0,
            "Inhibit_time": 300,
            "Data_Items": [{
                  "Description": "Systemstatus-Bitset",
                  "PV_name": "SysStatus_Tel0",
                  "type": "8Bit"
               },
               {
                  "Description": "IO-Bitset",
                  "PV_name": "IOStatus",
                  "type": "8Bit"
               },
               {
                  "Alias": "AI0_Temp",
                  "Description": "Temperatur Fühler 0",
                  "PV_name": "Analog_In_IO0",
                  "PV_at_data_zero": -30,
                  "PV_at_data_max": 97.5,
                  "Trigger_delta": 4,
                  "type": "8Bit",
                  "Unit": "°C"
               },
               {
                  "Description": "PLMulti-II Spannung L1--N Mittelwert / Fehlercode
  Modbus",
                  "PV_name": "U_L1_m",
                  "PV_at_data_max": 409.5,
                  "Valid_max": 400,
```

```
                "Trigger_delta": 2,
                "type": "12Bit",
                "Unit": "V"
            },
            {
                "Description": "PLMulti-II Spannung L2--N Mittelwert",
                "PV_name": "U_L2_m",
                "PV_at_data_max": 409.5,
                "Valid_max": 400,
                "Trigger_delta": 2,
                "type": "12Bit",
                "Unit": "V"
            },
            {
                "Description": "PLMulti-II Spannung L3--N Mittelwert",
                "PV_name": "U_L3_m",
                "PV_at_data_max": 409.5,
                "Valid_max": 400,
                "Trigger_delta": 2,
                "type": "12Bit",
                "Unit": "V"
            },
            {
                "Description": "PLMulti-II Strom L1 Mittelwert",
                "PV_name": "I_feed_L1_m",
                "PV_at_data_zero": -1638,
                "PV_at_data_max": 1638,
                "Valid_min": -1250,
                "Valid_max": 1250,
                "Trigger_delta": 10,
                "type": "12Bit",
                "Unit": "A"
            },
            {
                "Description": "PLMulti-II Strom L2 Mittelwert",
                "PV_name": "I_feed_L2_m",
                "PV_at_data_zero": -1638,
                "PV_at_data_max": 1638,
                "Valid_min": -1250,
                "Valid_max": 1250,
                "Trigger_delta": 10,
                "type": "12Bit",
                "Unit": "A"
            },
            {
                "Description": "PLMulti-II Strom L3 Mittelwert",
                "PV_name": "I_feed_L3_m",
                "PV_at_data_zero": -1638,
                "PV_at_data_max": 1638,
```

```
                    "Valid_min": -1250,
                    "Valid_max": 1250,
                    "Trigger_delta": 10,
                    "type": "12Bit",
                    "Unit": "A"
                },
                {
                    "Description": "PLMulti-II Wirkleistung L1 Einspeisung Mittelwert",
                    "PV_name": "P_feed_L1_m",
                    "PV_at_data_zero": -327.6,
                    "PV_at_data_max": 327.6,
                    "Valid_min": -327.6,
                    "Valid_max": 327.6,
                    "Trigger_delta": 10,
                    "type": "12Bit",
                    "Unit": "kW"
                },
                {
                    "Description": "PLMulti-II Wirkleistung L2 Einspeisung Mittelwert",
                    "PV_name": "P_feed_L2_m",
                    "PV_at_data_zero": -327.6,
                    "PV_at_data_max": 327.6,
                    "Valid_min": -327.6,
                    "Valid_max": 327.6,
                    "Trigger_delta": 10,
                    "type": "12Bit",
                    "Unit": "kW"
                },
                {
                    "Description": "PLMulti-II Wirkleistung L3 Einspeisung Mittelwert",
                    "PV_name": "P_feed_L3_m",
                    "PV_at_data_zero": -327.6,
                    "PV_at_data_max": 327.6,
                    "Valid_min": -327.6,
                    "Valid_max": 327.6,
                    "Trigger_delta": 10,
                    "type": "12Bit",
                    "Unit": "kW"
                },
                {
                    "Description": "PLMulti-II Gesamtverzerrung Spannung L1, L2, L3 ge-
  mittelt / Fehlercode Modbus",
                    "PV_name": "THD_U123_m",
                    "PV_at_data_max": 102.375,
                    "Valid_max": 100,
                    "Trigger_delta": 5,
                    "type": "12Bit",
                    "Unit": "%"
                }
```

```
281             ],
282             "Calc_Items": [{
283                   "Name": "S_L1_m",
284                   "Expr": "(U_L1_m * Math.abs(I_feed_L1_m))/1000.0",
285                   "Unit": "kVA"
286                },
287                {
288                   "Name": "S_L2_m",
289                   "Expr": "(U_L2_m * Math.abs(I_feed_L2_m))/1000.0",
290                   "Unit": "kVA"
291                },
292                {
293                   "Name": "S_L3_m",
294                   "Expr": "(U_L3_m * Math.abs(I_feed_L3_m))/1000.0",
295                   "Unit": "kVA"
296                },
297                {
298                   "Name": "Q_L1_m",
299                   "Expr": "Math.sqrt(Math.pow(S_L1_m,2) - Math.pow(P_feed_L1_m,2))",
300                   "Unit": "kVAr"
301                },
302                            {
303                   "Name": "Q_L2_m",
304                   "Expr": "Math.sqrt(Math.pow(S_L2_m,2) - Math.pow(P_feed_L2_m,2))",
305                   "Unit": "kVAr"
306                },
307                            {
308                   "Name": "Q_L3_m",
309                   "Expr": "Math.sqrt(Math.pow(S_L3_m,2) - Math.pow(P_feed_L2_m,2))",
310                   "Unit": "kVAr"
311                },
312                {
313                   "Name": "PF_L1_m",
314                   "Expr": "Math.abs(P_feed_L1_m / S_L1_m)",
315                   "Unit": ""
316                },
317                {
318                   "Name": "PF_L2_m",
319                   "Expr": "Math.abs(P_feed_L2_m / S_L2_m)",
320                   "Unit": ""
321                },
322                {
323                   "Name": "PF_L3_m",
324                   "Expr": "Math.abs(P_feed_L3_m / S_L3_m)",
325                   "Unit": ""
326                }
327             ]
328          },
329          {
```

```
330             "TelegramType": "Typ B",
331             "Description": "Telegramm wird Verletzung von Spannungs- / Stromgrenzwerten
  gesendet",
332             "Priority": true,
333             "LoRaWAN_PortNo": 21,
334             "Event_time": 0,
335             "Inhibit_time": 300,
336             "Data_Items": [{
337                  "Description": "Momentanwert Spannung L1--N",
338                  "PV_name": "U_L1",
339                  "PV_at_data_max": 655.35,
340                  "Valid_max": 400,
341                  "Trigger_delta": 2,
342                  "type": "16Bit",
343                  "Unit": "V"
344               },
345               {
346                  "Description": "Momentanwert Spannung L2--N",
347                  "PV_name": "U_L2",
348                  "PV_at_data_max": 655.35,
349                  "Valid_max": 400,
350                  "Trigger_delta": 2,
351                  "type": "16Bit",
352                  "Unit": "V"
353               },
354               {
355                  "Description": "Momentanwert Spannung L3--N",
356                  "PV_name": "U_L3",
357                  "PV_at_data_max": 655.35,
358                  "Valid_max": 400,
359                  "Trigger_delta": 2,
360                  "type": "16Bit",
361                  "Unit": "V"
362               },
363               {
364                  "Description": "Momentanwert Strom Einspeisung L1",
365                  "PV_name": "I_feed_L1",
366                  "PV_at_data_zero": -1310.7,
367                  "PV_at_data_max": 1310.7,
368                  "Valid_min": -1250,
369                  "Valid_max": 1250,
370                  "Trigger_delta": 5,
371                  "type": "16Bit",
372                  "Unit": "A"
373               },
374               {
375                  "Description": "Momentanwert Strom Einspeisung L2",
376                  "PV_name": "I_feed_L2",
377                  "PV_at_data_zero": -1310.7,
```

```
                    "PV_at_data_max": 1310.7,
                    "Valid_min": -1250,
                    "Valid_max": 1250,
                    "Trigger_delta": 5,
                    "type": "16Bit",
                    "Unit": "A"
                },
                {
                    "Description": "Momentanwert Strom Einspeisung L3",
                    "PV_name": "I_feed_L3",
                    "PV_at_data_zero": -1310.7,
                    "PV_at_data_max": 1310.7,
                    "Valid_min": -1250,
                    "Valid_max": 1250,
                    "Trigger_delta": 5,
                    "type": "16Bit",
                    "Unit": "A"
                },
                {
                    "Description": "Systemstatus-Bitset",
                    "PV_name": "SysStatus_Tel123",
                    "type": "8Bit"
                }
            ]
        },
        {
            "TelegramType": "Typ C",
            "Description": "Telegramm wird bei Verletzung von Leistungsgrenzwerten ge-
  sendet",
            "Priority": true,
            "LoRaWAN_PortNo": 22,
            "Event_time": 0,
            "Inhibit_time": 300,
            "Data_Items": [{
                    "Description": "Momentanwert Wirkleistung Einspeisung L1",
                    "PV_name": "P_feed_L1",
                    "PV_at_data_zero": -655.35,
                    "PV_at_data_max": 655.35,
                    "Valid_min": -500,
                    "Valid_max": 500,
                    "Trigger_delta": 10,
                    "type": "16Bit",
                    "Unit": "kW"
                },
                {
                    "Description": "Momentanwert Wirkleistung Einspeisung L2",
                    "PV_name": "P_feed_L2",
                    "PV_at_data_zero": -655.35,
                    "PV_at_data_max": 655.35,
```

```
                "Valid_min": -500,
                "Valid_max": 500,
                "Trigger_delta": 10,
                "type": "16Bit",
                "Unit": "kW"
            },
            {
                "Description": "Momentanwert Wirkleistung Einspeisung L3",
                "PV_name": "P_feed_L3",
                "PV_at_data_zero": -655.35,
                "PV_at_data_max": 655.35,
                "Valid_min": -500,
                "Valid_max": 500,
                "Trigger_delta": 10,
                "type": "16Bit",
                "Unit": "kW"
            },
            {
                "Description": "Momentanwert Scheinleistung Einspeisung L1",
                "PV_name": "S_feed_L1",
                "PV_at_data_max": 655.35,
                "Valid_max": 500,
                "Trigger_delta": 10,
                "type": "16Bit",
                "Unit": "kVA"
            },
            {
                "Description": "Momentanwert Scheinleistung Einspeisung L2",
                "PV_name": "S_feed_L2",
                "PV_at_data_max": 655.35,
                "Valid_max": 500,
                "Trigger_delta": 10,
                "type": "16Bit",
                "Unit": "kVA"
            },
            {
                "Description": "Momentanwert Scheinleistung Einspeisung L3",
                "PV_name": "S_feed_L3",
                "PV_at_data_max": 655.35,
                "Valid_max": 500,
                "Trigger_delta": 10,
                "type": "16Bit",
                "Unit": "kVA"
            },
            {
                "Description": "Systemstatus-Bitset",
                "PV_name": "SysStatus_Tel123",
                "type": "8Bit"
            }
```

```
475             ],
476             "Calc_Items": []
477          },
478          {
479             "TelegramType": "Typ D",
480             "Description": "Telegramm wird bei Überschreitung von THD- Grenzwerten ge-
   sendet",
481             "Priority": true,
482             "LoRaWAN_PortNo": 23,
483             "Event_time": 0,
484             "Inhibit_time": 300,
485             "Data_Items": [{
486                   "Description": "Momentanwert Gesamtverzerrung Spannung L1",
487                   "PV_name": "THD_U_L1",
488                   "PV_at_data_max": 255,
489                   "Valid_max": 100,
490                   "Trigger_delta": 2,
491                   "type": "8Bit",
492                   "Unit": "%"
493                },
494                {
495                   "Description": "Momentanwert Gesamtverzerrung Spannung L2",
496                   "PV_name": "THD_U_L2",
497                   "PV_at_data_max": 255,
498                   "Valid_max": 100,
499                   "Trigger_delta": 2,
500                   "type": "8Bit",
501                   "Unit": "%"
502                },
503                {
504                   "Description": "Momentanwert Gesamtverzerrung Spannung L3",
505                   "PV_name": "THD_U_L3",
506                   "PV_at_data_max": 255,
507                   "Valid_max": 100,
508                   "Trigger_delta": 2,
509                   "type": "8Bit",
510                   "Unit": "%"
511                },
512                {
513                   "Description": "Momentanwert Gesamtverzerrung Strom Einspeisung L1",
514                   "PV_name": "THD_I_feed_L1",
515                   "PV_at_data_max": 255,
516                   "Valid_max": 100,
517                   "Trigger_delta": 2,
518                   "type": "8Bit",
519                   "Unit": "%"
520                },
521                {
522                   "Description": "Momentanwert Gesamtverzerrung Strom Einspeisung L2",
```

```
                    "PV_name": "THD_I_feed_L2",
                    "PV_at_data_max": 255,
                    "Valid_max": 100,
                    "Trigger_delta": 2,
                    "type": "8Bit",
                    "Unit": "%"
                },
                {
                    "Description": "Momentanwert Gesamtverzerrung Strom Einspeisung L3",
                    "PV_name": "THD_I_feed_L3",
                    "PV_at_data_max": 255,
                    "Valid_max": 100,
                    "Trigger_delta": 2,
                    "type": "8Bit",
                    "Unit": "%"
                },
                {
                    "Description": "Systemstatus-Bitset",
                    "PV_name": "SysStatus_Tel123",
                    "type": "8Bit"
                }
            ]
        }
    ]
},


"SN991000001": {
    "LoRaWAN_Keys": {
        "appeui": ["70B3001122334455"],
        "deveui": ["AABBCCD991000001"],
        "appkey": ["00112233445566778899AABBCCDDEEFF"]
    }
},

"SN991000002": {
    "LoRaWAN_Keys": {
        "appeui": ["70B3D57EF0003857"],
        "deveui": ["AABBCCD991000002"],
        "appkey": ["FFEEDDCCBBAA99887766554433221100"]
    }
}
}
```