

**Projekt: solaranzeige.de**

**Eigene Erweiterungen der Solaranzeige mit Hilfe der `_math`  
Datei schreiben.**

**Update Sicherheit herstellen.**

**Stand Februar 2024**

**Dokument EE017**

## **Inhaltsverzeichnis**

Übersicht:.....	2
Was ist zu tun und was hat es mit der ' <code>_math</code> ' Datei auf sich?.....	2
Welche Variablen stehen in der Erweiterung zur Verfügung?.....	3
Zusätzliche Daten in die Datenbank schreiben:.....	4
Auslesen der Influx Datenbank:.....	5
Was hat es mit der ' <code>user_init.php</code> ' auf sich?.....	6
Mehrere gleiche Geräte:.....	6
Fehleranalyse:.....	7
Eine Übung: Temperatur des Raspberry abspeichern:.....	7
Was hat es mit der API auf sich?.....	8

## Übersicht:

Ab der Image Version 4.7.2 ist es möglich, eigene Erweiterungen für die einzelnen auszulesenden Geräten zu schreiben, ohne dass diese bei einem Update überschrieben werden. So können eigene Berechnungen angestellt und mehr Datenpunkte abgefragt werden oder zusätzliche Datenbankabfragen integriert und auch zusätzliche Daten abgespeichert werden. Fast alles ist möglich, **wenn man PHP programmieren kann**. Auch ist es so möglich Python Skripte mit zu verarbeiten. Bitte nach „PHP and Python Integration“ oder „Python in PHP integrieren“ googlen.

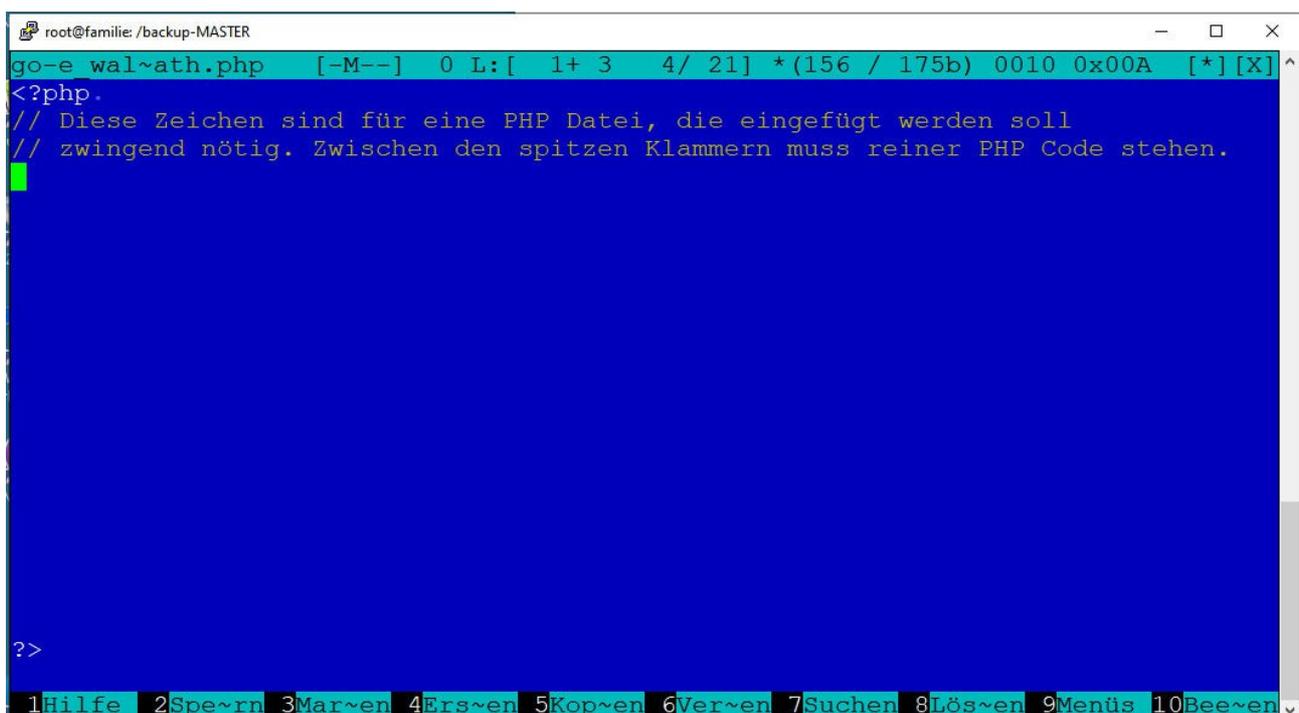
Dort sind Tutorials, wie das geht, zu finden. Jede Erweiterung bezieht sich auf genau eine Geräteklasse. Die Erweiterungen sind auch austauschbar, wenn man darauf bei dem Erstellen achtet. So kann man auch Erweiterungen für Andere schreiben. Bitte darauf achten dass es sich um wirkliche PHP Dateien handelt. Also mit den Zeichen „**<?php**“ beginnend und mit „**?>**“ endend. Ansonsten baut man sich ungewollt eine Sicherheitslücke ein. Funktionieren würde es trotzdem. Wer keine Programmierkenntnisse hat, wird mit dieser Beschreibung nicht viel anfangen können.

## Was ist zu tun und was hat es mit der '\_math' Datei auf sich?

( \_math Datei => mathematische Berechnungen )

Jedes Gerät wird mit einem eigenen Script ausgelesen. Die Verbindung zwischen Regler Nummer und Script Name findet man in der Datei „regler\_auslesen.php“ Nehmen wir einmal an, wir haben eine go-eCharger Wallbox. Die hat die Regler Nummer 29. In der „regler\_auslesen.php“ Datei finden wir Regler 29 in Verbindung mit der Datei „go-e\_wallbox.php“

Möchte ich jetzt eine Erweiterung für dieses Gerät selber schreiben, muss ich eine Datei mit dem Namen „go-e\_wallbox\_math.php“, in dem Verzeichnis /var/www/html/ neu anlegen. Diese Datei muss eine PHP Datei sein, die mindestens folgende Zeilen hat:



```
root@familie: /backup-MASTER
go-e_wal~ath.php [-M--] 0 L:[ 1+ 3 4/ 21] *(156 / 175b) 0010 0x00A [*][X] ^
<?php.
// Diese Zeichen sind für eine PHP Datei, die eingefügt werden soll
// zwingend nötig. Zwischen den spitzen Klammern muss reiner PHP Code stehen.
?>
1Hilfe 2Spe~rn 3Mar~en 4Ers~en 5Kop~en 6Ver~en 7Suchen 8Lös~en 9Menüs 10Bee~en
```

Die Dateien, in denen die eigene PHP Software eingefügt wird, muss immer den Zusatz „\_math“ haben. Heißt der Script zum Auslesen des Huawei SmartMeter „huawei\_SL.php“ so muss die eigene Datei „huawei\_SL\_math.php“ heißen. Bei dem Script „victron\_solarregler.php“ folglich „victron\_solarregler\_math.php“

Ist so eine Datei im Verzeichnis /var/www/html/ vorhanden, wird sie im Programm, nach dem Auslesen des Gerätes, jedoch vor dem Abspeichern der Daten, durchlaufen. Ist sie nicht vorhanden, passiert nichts. Möchte man auf das Abspeichern der Daten Einfluss nehmen, dann so eine Datei erstellen und den eigenen PHP Code darin platzieren. Auf diese Weise können auch Python Scripte inkludiert werden. Wie das genau geht steht im Internet. Die eigene Datei wird genau da verarbeitet, wo das Auslesen des Gerätes beendet wird und die Daten dann per MQTT eventuell versendet werden. Danach werden die Daten in die Datenbank abgespeichert. Die Daten werden immer in dieser Reihenfolge verarbeitet.

1. Falls das Gerät dafür vorbereitet ist, werden Befehle zum Gerät gesendet.
2. Die Daten werden aus dem Gerät ausgelesen.
3. Hier werden die eigenen Erweiterungen verarbeitet.
4. Die Daten werden, falls gewünscht per MQTT zum nächsten Brocker gesendet.
5. Die Daten werden zur entfernten Datenbank gesendet. (Falls gewünscht)
6. Die Daten werden in die lokale Datenbank geschrieben.
7. Falls gewünscht werden Daten zur HomeMatic gesendet.
8. Falls nötig werden Nachrichten über Pushover gesendet.

## Welche Variablen stehen in der Erweiterung zur Verfügung?

1. \$aktuelleDaten[] Array der aktuellen ausgelesenen Daten
2. \$Tracelevel [ 1 – 10 ] normal = 7
3. \$Homematic true / false
4. \$Messenger true / false
5. \$MQTT true / false
6. \$Pfad Pfad zum Verzeichnis der Dateien
7. \$RaspiTemp Raspberry Temperatur in °C
8. \$mra\_i welche x.user.config.php wird gerade verarbeitet. (Multi-Regler)
9. \$zentralerTimestamp (Multi-Regler-Version)

Im Prinzip alle Variablen der jeweiligen x.user.config.php und das Array \$aktuelleDaten in dem die ausgelesenen Werte des Gerätes stehen. Das Array hat je nach Gerät immer einen anderen Aufbau. Am besten das Array in die LOG Datei schreiben lassen, dann kann man sich den Aufbau genau ansehen. Mit dieser Zeile wird das Array in die LOG Datei geschrieben.

```
$funktionen->log_schreiben(print_r($aktuelleDaten,1)," ",1);
```

## Zusätzliche Daten in die Datenbank schreiben:

Möchte man zusätzliche Daten in die Datenbank schreiben, dann geht das so:

Man kann zu den bestehenden Measurement's Felder hinzu fügen oder auch eigene Measurement's mit Feldern erzeugen. Die nötige Query muss folgenden Aufbau haben:

```
Measurement    Feldname=Wert    zentralerTimestamp
```

Beispiel:

```
$aktuelleDaten['ZusatzQuery'] = 'Summen NeuerFeldname=Wert1 '.$aktuelleDaten['zentralerTimestamp'];
```

oder auch

```
$aktuelleDaten['ZusatzQuery'] = 'MeasurementNeu NeuerFeldname1=Wert, NeuerFeldname2=Wert '.$aktuelleDaten['zentralerTimestamp'];
```

(Alles in einer Zeile)

Möchte man 2 Measurements neu schreiben, dann muss ein 'New Line' Zeichen dazwischen „\n“  
Die gesamte Query muss in die Variable \$aktuelleDaten['ZusatzQuery']

Die Zusatz Query kann mehrere Measurements sowie mehrere Feldnamen enthalten. Sie darf jedoch keine schon existierende Feldnamen enthalten, da die ja kurz danach von dem normalen Script eingefügt werden. Bitte Fragen dazu im Forum stellen. Die Reihenfolge der Measurement's ist nicht wichtig. Darauf muss man nicht achten. Bitte kurz bevor man mit den eigenen Erweiterungen anfängt noch einmal ein Update machen, da sich diese Funktionen noch in der Entwicklung befinden.

Hier bietet es sich natürlich auch an, die Daten zusätzlich in eine 3. Influx noch zu schreiben oder in eine andere Datenbank wie z.B. eine MySQL oder SQLite. Je nachdem in welchem Format man die Daten benutzen möchte.

Auch könnte man die Datenbankstruktur vollkommen ändern. Es bieten sich viele weitere Möglichkeiten an.

Beispiel, wie man Daten in 2 unterschiedliche Measurements schreiben kann, auch wenn diese schon vorhanden sind!

```
$aktuelleDaten["ZusatzQuery"] = "Measurement1 Erzeugung=".$Variable1;
$aktuelleDaten["ZusatzQuery"] .= ",Einspeisung=".$Variable2;
$aktuelleDaten["ZusatzQuery"] .= ",ErzeugungTag=".$Variable3;
$aktuelleDaten["ZusatzQuery"] .= "    ".$aktuelleDaten["zentralerTimestamp"];
$aktuelleDaten["ZusatzQuery"] .= "\n";
$aktuelleDaten["ZusatzQuery"] = "Measurement2 EinspeisungTag=".$Variable4;
$aktuelleDaten["ZusatzQuery"] .= ",Netzbezug=".$Variable5;
$aktuelleDaten["ZusatzQuery"] .= "    ".$aktuelleDaten["zentralerTimestamp"];
$aktuelleDaten["ZusatzQuery"] .= "\n";
```

Wichtig ist die Leerstelle nach dem Measurement Namen und das Zeichen '\n' nach dem Timestamp. Mehrere Variablen werden mit Komma getrennt. Zwischen den Variablen und dem Timestamp muss eine Leerstelle.

'.=' heißt Zeile anhängen.

## Auslesen der Influx Datenbank:

Hier ist ein CODE Schnipsel, wie man eine Datenbank auslesen und die Werte weiterverarbeiten kann. In diesem Fall geht es um die Datenbank „solaranzeige“ mit dem Measurement „PV“

```
$Datenbank = "solaranzeige";
$Measurement = "PV";
$sql = urlencode('select * from '.$Measurement.' order by time desc limit 1');
$ch = curl_init('http://localhost/query?db='.$Datenbank.'&precision=s&q='.$sql);

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, "POST");
curl_setopt($ch, CURLOPT_TIMEOUT, 15);
curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 12);
curl_setopt($ch, CURLOPT_PORT, 8086);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

$Ergebnis["result"] = curl_exec($ch);
$Ergebnis["rc_info"] = curl_getinfo ($ch);
$Ergebnis["JSON_Ausgabe"] = json_decode($Ergebnis["result"],true,10);
$Ergebnis["errorno"] = curl_errno($ch);

if ($Ergebnis["rc_info"]["http_code"] == 200 or $Ergebnis["rc_info"]["http_code"] == 204)
{
    $Ergebnis["Ausgabe"] = true;
}

curl_close($ch);
unset($ch);
```

```

if (!isset($Ergebnis["JSON_Ausgabe"]["results"][0]["series"])) {
    log_schreiben("Es fehlt die Datenbank solaranzeige mit dem Measurement PV oder sie ist
leer.", "|- ", 3);
}
else {
    for ($h = 1; $h < count($Ergebnis["JSON_Ausgabe"]["results"][0]["series"][0]
["columns"]); $h++) {
        $DB1[$Ergebnis["JSON_Ausgabe"]["results"][0]["series"][0]["columns"][$h]] =
$Ergebnis["JSON_Ausgabe"]["results"][0]["series"][0]["values"][0][$h];
    }
    log_schreiben("Datenbank: solaranzeige DB1 ".print_r($DB1,1), " ", 10);
}

```

## Was hat es mit der 'user\_init.php' auf sich?

So wie man die „\_math“ Datei anlegen kann, so kann man auch eine PHP Datei „user\_init.php“ erstellen. Diese Datei wird *einmal* beim Booten durchlaufen. In dieser Datei kann man zum Beispiel Schnittstellen initialisieren. Da die Datei am Ende des boot Vorgangs durchlaufen wird, kann man z.B. die USB Schnittstellen anders initialisieren, wie das Solaranzeigen Programm. Wird ein Wechselrichter per serieller Schnittstelle /dev/ttyUSB0 mit 9600 Baud ausgelesen und man möchte die serielle Geschwindigkeit jedoch mit 19200 Baud betreiben, so kann man die serielle Schnittstelle /dev/ttyUSB0 mit 19200 Baud in dieser user\_init.php umprogrammieren. Wenn im Wechselrichter die Schnittstelle auch auf 19200 Baud eingestellt ist, sollte es funktionieren.

Alles was in der usb\_init.php bzw. multi\_usb\_init.php eingestellt wird und weitere Schnittstellen. Kann in der user\_init.php noch einmal abgeändert / ergänzt werden, wenn die Datei existiert. Bei der Multi-Regler-Version ist alles etwas komplizierter, aber möglich.

## Mehrere gleiche Geräte:

Es kann vorkommen, dass man mehrere gleiche Geräte mit einer Multi-Regler-Version ausließt. Dann würde die \_math Datei ja mehrmals durchlaufen werden. Das ist richtig, anhand der Variable \$GeraeteNummer kann ich jedoch erkennen, welches Gerät mit welcher x.user.config.php gerade ausgelesen wird und kann die Berechnung so auf ein Gerät begrenzen, auch wenn es mehrere gleiche Geräte davon gibt. Die \$GeraeteNummer ist immer gleich mit der Nummer der user.config.php. ( x.user.config.php ) So kann man die gleiche \_math Datei für mehrere Geräte des gleichen Typs benutzen.

Eine \_math Datei zu schreiben bedeutet aber, das man PHP oder Python Programmieren kann. Das Herauskopieren aus anderen Dateien, ohne Programmierkenntnisse, wird scheitern.

## Fehleranalyse:

Hat man PHP Fehler in dem eigenen Script, läuft das gesamte Programm nicht mehr. Fehler kann man in der LOG Datei `/var/www/log/php.log` sehen. Funktioniert die gesamte solaranzeige nicht mehr, dann zu erst in die `php.log` schauen. Läuft das Programm, aber der Output ist nicht so wie erwartet, dann bitte in die LOG Datei `/var/www/log/solaranzeige.log` schauen.

Um Fehler auf zu spüren den Tracelevel auf 8 oder 9 erhöhen. Normal ist 7. Möchte man wenig Logeinträge dann auf 5 reduzieren.

## Eine Übung: Temperatur des Raspberry abspeichern:

Eine schöne Übung für den Anfang ist, die Temperatur des Raspberry's in der Influx Datenbank im Measurement „Service“ abzuspeichern. Natürlich kann man die Daten auch in jedem anderen oder auch neuen Measurement abspeichern. Dazu muss man eine „\_math“ Datei, wie vorher beschrieben, erstellen und dort folgende Einträge machen:

```
/*
// Raspberry Temperatur in die Influx Datenbank speichern
// Die Temperatur steckt in der Variable $RaspiTemp
*/

// So wird die Zusatz Query zusammengestellt.
// Alle Daten werden in die aktuelle Datenbank des Gerätes in das Measurement "Service" geschrieben
// Der Zeitstempel ist der 'zentrale Timestamp'
// Damit ist die Visualisierung in Grafana sehr einfach.

$aktuelleDaten["ZusatzQuery"] = "Service RaspiTemp=".round($RaspiTemp,1);
$aktuelleDaten["ZusatzQuery"] .= " ".$aktuelleDaten["zentralerTimestamp"];

// Wenn der Wert auch in die LOG Datei geschrieben werden soll.

$funktionen->log_schreiben("Raspberry Temperatur: ".round($RaspiTemp,1)." °C", "> ",5);
```

In der Variable „\$RaspiTemp“ steht die Temperatur. Diese wird über die ZusatzQuery in die Datenbank geschrieben und kann so bequem im Dashboard angezeigt werden. Die „\_math“ Datei kann natürlich zusätzlich noch für andere Berechnungen genutzt werden.

Es gibt noch eine 2. Variable die man abspeichern kann: \$FreierSpeicher  
Dort steht der noch freie Speicher, der auf der SD-Karte / USB-Stick noch zur Verfügung steht. Als 2. Übung kann man diesen auch in die Datenbank abspeichern und dann im Dashboard anzeigen.

```
/*
// Raspberry Temperatur in die Influx Datenbank speichern
// Die Temperatur steckt in der Variable $RaspiTemp
*/

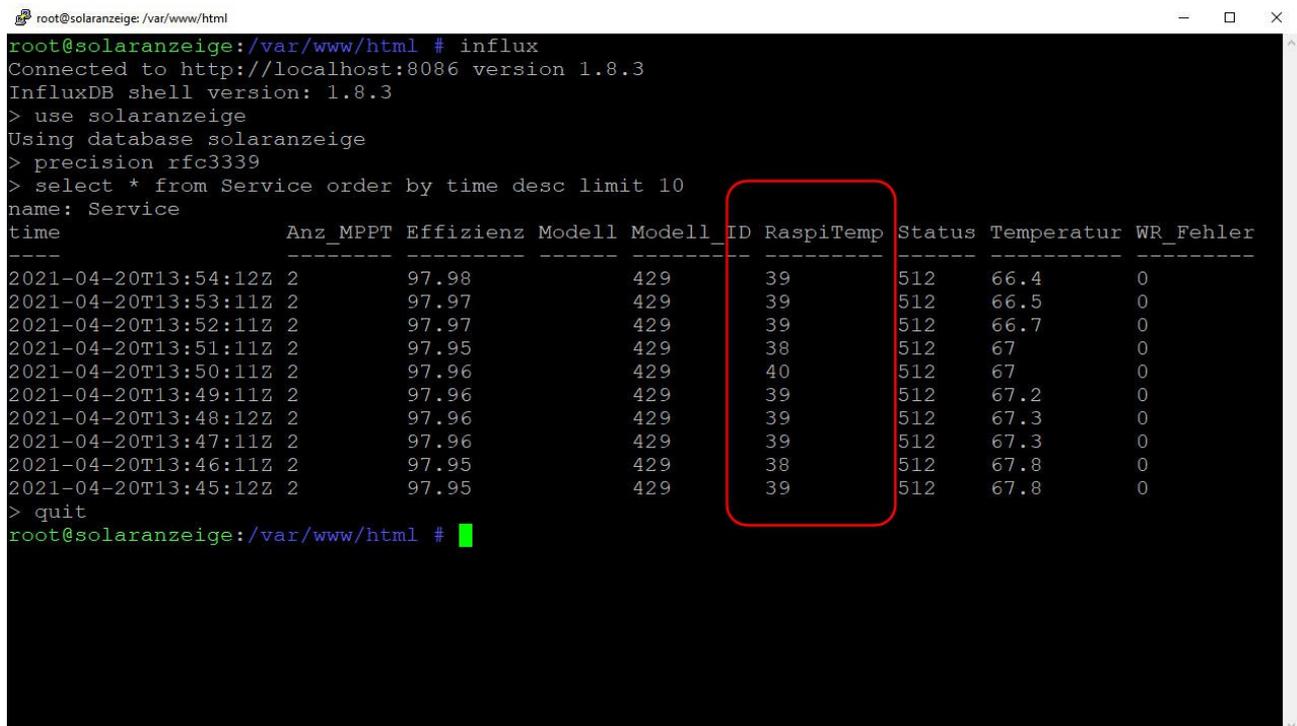
// So wird die Zusatz Query zusammengestellt.
// Alle Daten werden in die aktuelle Datenbank des Gerätes in das Measurement "Service" geschrieben
// Der Zeitstempel ist der 'zentraler Timestamp'
// Damit ist die Visualisierung in Grafana sehr einfach.

$aktuelleDaten["ZusatzQuery"] = "Service RaspiTemp=".round($RaspiTemp,1);
$aktuelleDaten["ZusatzQuery"] .= ",RaspiFreierSpeicher=\"".$FreierSpeicher."\"";
$aktuelleDaten["ZusatzQuery"] .= " ".$zentralerTimestamp;
```

Möchte man prüfen, ob die Daten wirklich in die InfluxDB geschrieben werden, dann auf der Konsole folgende Eingaben machen:

```
influx
use solaranzeige
precision rfc3339
select * from Service order by time desc limit 10
quit
```

Jetzt sollte man in der Spalte „RaspiTemp“ die Temperatur in Grad sehen.



```
root@solaranzeige: /var/www/html
root@solaranzeige: /var/www/html # influx
Connected to http://localhost:8086 version 1.8.3
InfluxDB shell version: 1.8.3
> use solaranzeige
Using database solaranzeige
> precision rfc3339
> select * from Service order by time desc limit 10
name: Service
time                Anz_MPPT  Effizienz  Modell  Modell_ID  RaspiTemp  Status  Temperatur  WR_Fehler
-----
2021-04-20T13:54:12Z 2          97.98      429     429        39         512     66.4        0
2021-04-20T13:53:11Z 2          97.97      429     429        39         512     66.5        0
2021-04-20T13:52:11Z 2          97.97      429     429        39         512     66.7        0
2021-04-20T13:51:11Z 2          97.95      429     429        38         512     67         0
2021-04-20T13:50:11Z 2          97.96      429     429        40         512     67         0
2021-04-20T13:49:11Z 2          97.96      429     429        39         512     67.2        0
2021-04-20T13:48:12Z 2          97.96      429     429        39         512     67.3        0
2021-04-20T13:47:11Z 2          97.96      429     429        39         512     67.3        0
2021-04-20T13:46:11Z 2          97.95      429     429        38         512     67.8        0
2021-04-20T13:45:12Z 2          97.95      429     429        39         512     67.8        0
> quit
root@solaranzeige: /var/www/html # █
```

## Was hat es mit der API auf sich?

Eine zusätzliche Möglichkeit ist, Daten in und aus der Influx Datenbank zu bekommen, die Nutzung der API. (*Application Programming Interface*, Programmierschnittstelle) Mit Hilfe der API kann man einzelne oder auch viele Daten in die vorhandenen Solaranzeigen Influx Datenbanken schreiben bzw. auslesen. Das Dokument dazu heißt API.pdf und befindet sich auf dem Support Forum [www.solaranzeige.de](http://www.solaranzeige.de)

Die Daten werden mittels eines XML Dokumentes in die Datenbanken geschrieben bzw. die zuletzt gespeicherten Werte ausgelesen. Das funktioniert aus vielen verschiedenen Programmiersprachen heraus.

© Solaranzeige.de Nachdruck und Verbreitung nur mit unserer schriftlichen Genehmigung.