# Fronius Solar API V0

EN

Operating Instructions

System monitoring

42,0410,2011 002-06082013

# Contents

# 1 Introduction

The Fronius Solar API is a means for third parties to obtain data from various Fronius devices (inverters, Sensor-Cards, StringControls) in a defined format through a central facility which acts as a proxy (e.g. Fronius Datalogger Web or Fronius Solar.web).

Currently, the only way to interact with this API is by making a HTTP request to a specific CGI. The URLs for the particular requests and the devices supporting them are listed at the beginning of each request description.
The API is versioned, meaning that multiple versions of this API may be available on the same device. The URLs in this document always point to the version of the API which this document describes. The highest supported version on the device can be queried. See 2.3.1 for details.
In order to check your product for compatibility with this version of the API specification, please see the separate document provided for this purpose.

Realtime requests will obtain the data directly from the devices and can therefore only be used when the devices are not in standby or unavailable in any other matter.

# 2 General Considerations

## 2.1 Output Formats

Currently, the only output format supported is JSON, a lightweight data interchange format. It is easy to read and write for both humans and machines and it offers some advantages over XML, like basic typing and a leaner structure.

## 2.2 Data Types

### 2.2.1 Numeric Types

JSON only knows one kind of numeric types, which can represent both floating point and integer values. It is however possible to specify a type in JSON description, but it is always in the hands of the interpreting system into which datatype a numeric node is converted.
Which range a certain numeric node actually can have is often determined by the device providing the value, and may also vary depending on the type of device (e.g. "UAC" can be an integer value on older inverters, but a floating point value on newer ones).

This means we cannot reliably specify value ranges for all requests. So it is the responsibility of the API user to determine whether a value fits into a certain datatype in his language of choice.
What we can do is to specify whether a certain value is a floating point value (markes as "number") or an integer value (marked as "integer"), where "integer" must not be interpreted as the datatype "int" like available in C/C++, it just means it is a value without decimal places.
For these specifications, please refer to the sections discussing the respective request.

### 2.2.2 Date/Time

Information on date/time is always (and can only be) represented by a string. The format for these strings inside this API has been defined as follows.

- Strings which include information on both date and time are always in RFC3339 format with time zone offset or Zulu marker.
  See Section 5.6 of RFC3339
  Example 1: `2011-10-20T10:23:17+02:00` (UTC+2)
  Example 2: `2011-10-20T08:23:17Z` (UTC)

- Strings which only include information on the date are of the format `yyyy-MM-dd`.
- Strings which only include information on the time are of the format `hh:mm:ss`.
- If no information on the time zone is given, any date/time specification is considered to be in local time of the PV system.

## 2.3 Requests

Currently, the only request protocol supported is HTTP.

### 2.3.1 Querying of API version

Future versions of the API export the highest supported version on the device using the URL `/solar_api/GetAPIVersion.cgi`.

Listing 1: Object structure of GetAPIVersion response

```
object {

    # Numeric version of the API.
    number APIVersion;

    # URL under which the CGIs for the requests can be reached.
    string BaseURL;
}
```

Listing 2: Example: Complete response for GetAPIVersion request

```
{
    "APIVersion" : 1,
    "BaseURL" : "/solar_api/v1/"
}
```

The first version of the API does not support this mechanism and returns HTTP error 404 instead. So any user that wants to support different versions of the API shall assume the following response if the `/solar_api/GetAPIVersion.cgi` fails with HTTP status code 404:

Listing 3: Assumed response for HTTP status code 404

```
{
    "APIVersion" : 0,
    "BaseURL" : "/solar_api/"
}
```

## 2.4 Responses

The response will always be a valid JSON string ready to be evaluated by standard libraries.
If the response is delivered through HTTP, the `Content-Type` Header shall be either `text/javascript` or `application/json`.

All JSON structures are described using *Orderly JSON*, a textual format for describing JSON data. Please refer to the online documentation on *Orderly* for details.
Note that the definitions of some response bodies are not totally accurate, because there's no (known) way to express nodes named after values/channels (e.g. objects which are named "PAC" or "Power"). But each description is accompanied by an example which should clear up any uncertainty.

The contents of the response object will vary depending on the preceding request but it always contains a common response header and a request body.

Listing 4: Object structure of valid response

```
object {

  object Head: {}*;

  object Body: {}*;

}
```

Listing 5: Example: Complete response for GetInverterRealtimeData request


```
{
    "Head": {
        "RequestArguments": {
             "Scope": "Device",
             "DeviceIndex": 0,
             "DataCollection": "CommonInverterData"
         },
        "Status": {
            "Code": 0,
            "Reason": "",
            "UserMessage": ""
        },
        "Timestamp": "2011-10-20T10:09:14+02:00"
    },
    "Body": {
        "Data" : {
            "DAY_ENERGY" : {
                    "Value" : 8000,
                    "Unit" : "Wh"
            },
            "FAC" : {
                    "Value" : 50,
                    "Unit" : "Hz"
            },
            "IAC" : {
                    "Value" : 14.54,
                    "Unit" : "A"
            },
            "IDC" : {
                    "Value" : 8.2,
                    "Unit" : "A"
            },
            "PAC" : {
                    "Value" : 3373,
                    "Unit" : "W"
            },
            "SAC" : {
                    "Value" : 3413,
                    "Unit" : "VA"
            },
            "TOTAL_ENERGY" : {
                    "Value" : 45000,
                    "Unit" : "Wh"
            },
            "UAC" : {
                    "Value" : 232,
                    "Unit" : "V"
            },
            "UDC" : {
                    "Value" : 426,
                    "Unit" : "V"
            },
            "YEAR_ENERGY" : {
                    "Value" : 44000,
                    "Unit" : "Wh"
            },
            "DeviceStatus" : {
                    "DeviceState" : 7,
                    "MgmtTimerRemainingTime" : -1,
                    "ErrorCode" : 0,
                    "LEDColor" : 2,
                    "LEDState" : 0,
                    "StateToReset" : false
            }
        }
    }
```

```
}
```

### 2.4.1 Common Response Header

The common response header (CRH) is present in every response. It indicates, among other things, whether the request has been successful and the body of the response is valid.

Listing 6: Object Structure of Common Response Header

```
object {

  # Repetition of the parameters which produced this response.
  object {
    # Filled with properties named like the given parameters.
  }* RequestArguments;

  # Information about the response.
  object {

    # Indicates if the request went OK or gives a hint about what went wrong.
    # 0 means OK, any other value means something went wrong (e.g. Device not available,
    # invalid params, no data in logflash for given time, ...).
    integer Code;

    # Error message, may be empty.
    string Reason;

    # Error message to be displayed to the user, may be empty.
    string UserMessage;

  } Status;

  # RFC3339 timestamp in localtime of the datalogger.
  # This is the time the request was answered - NOT the time when the data
  # was queried from the device.
  string Timestamp;

};
```

### 2.4.2 Request Body

The request body contains the actual data produced by the request and is therefore different for each request. The object structures of the various response bodies will be detailed later in the description of the respective API request.

# 3 Realtime Requests

These requests will be provided where direct access to the realtime data of the devices is possible. This is currently the case for the Fronius Datalogger Web and the Fronius Datamanager.

In order to eliminate the need to specify each wanted value separately when making a request or querying each value separately, so called "Data Collections" were defined.
The values in these collections are gathered from one or more Fronius Solar Net messages and supplied to the user in a single response to a certain request.
It may be the case that more values are queried from the device than the user is interested in, but the overhead caused by these superfluous values should be negligible compared to the advantages this strategy provides for the user.

**If a device cannot provide some values of a DataCollection (e.g. because they are not implemented on the device) then those values are omitted from the response.**

## 3.1 *GetInverterRealtimeData* request

### 3.1.1 URL for HTTP requests

`/solar_api/GetInverterRealtimeData.cgi`

### 3.1.2 Parameters

| Parameter | Type | Range/Values/Pattern | Description |
|---|---|---|---|
| Scope | String | "Device"<br>"System" | Query specific device(s) or whole system |
| DeviceIndex | Integer | 0 . . . 99 | *Only needed for Scope "Device"*<br>Which inverter to query. |
| DataCollection | String | "CumulationInverterData"<br>"CommonInverterData"<br>"3PInverterData"<br>"MinMaxInverterData" | *Only needed for Scope "Device"*<br>Selects the collection of data that should be queried from the device.<br>See 3.1.3 for details. |

### 3.1.3 Data Collections

**CumulationInverterData** Values which are cumulated to generate a system overview.

| Value name | JSON type | Description |
|---|---|---|
| PAC | unsigned integer | AC power |
| DAY_ENERGY | unsigned integer | Energy generated on current day |
| YEAR_ENERGY | unsigned integer | Energy generated in current year |
| TOTAL_ENERGY | unsigned integer | Energy generated overall |
| DeviceStatus | object | Status information about inverter |

**CommonInverterData** Values which are provided by all types of Fronius inverters.

| Value name | JSON type | Description |
|---|---|---|
| PAC | unsigned integer | AC power |
| IAC | floating point | AC current |
| UAC | floating point | AC voltage |
| FAC | floating point | AC frequency |
| IDC | floating point | DC current |
| UDC | floating point | DC voltage |
| DAY_ENERGY | unsigned integer | Energy generated on current day |
| YEAR_ENERGY | unsigned integer | Energy generated in current year |
| TOTAL_ENERGY | unsigned integer | Energy generated overall |
| DeviceStatus | object | Status information about inverter |

**3PInverterData** Values which are provided by 3phase Fronius inverters.

| Value name | JSON type | Description |
|---|---|---|
| IAC_L1 | floating point | AC current Phase 1 |
| IAC_L2 | floating point | AC current Phase 2 |
| IAC_L3 | floating point | AC current Phase 3 |
| UAC_L1 | floating point | AC voltage Phase 1 |
| UAC_L2 | floating point | AC voltage Phase 2 |
| UAC_L3 | floating point | AC voltage Phase 3 |
| T_AMBIENT | signed integer | Ambient temperature |
| ROTATION_SPEED_FAN_FL | unsigned integer | Rotation speed of front left fan |
| ROTATION_SPEED_FAN_FR | unsigned integer | Rotation speed of front right fan |
| ROTATION_SPEED_FAN_BL | unsigned integer | Rotation speed of back left fan |
| ROTATION_SPEED_FAN_BR | unsigned integer | Rotation speed of back right fan |

**MinMaxInverterData** Minimum- and Maximum-values of various inverter values.

| Value name | JSON type | Description |
|---|---|---|
| DAY_PMAX | unsigned integer | Maximum AC power of current day |
| DAY_UACMAX | floating point | Maximum AC voltage of current day |
| DAY_UACMIN | floating point | Minimum AC voltage of current day |
| DAY_UDCMAX | floating point | Maximum DC voltage of current day |
| YEAR_PMAX | unsigned integer | Maximum AC power of current year |
| YEAR_UACMAX | floating point | Maximum AC voltage of current year |
| YEAR_UACMIN | floating point | Minimum AC voltage of current year |
| YEAR_UDCMAX | floating point | Maximum DC voltage of current year |
| TOTAL_PMAX | unsigned integer | Maximum AC power of current year |
| TOTAL_UACMAX | floating point | Maximum AC voltage overall |
| TOTAL_UACMIN | floating point | Minimum AC voltage overall |
| TOTAL_UDCMAX | floating point | Maximum DC voltage overall |

### 3.1.4 Object structure of request body (Scope "Device")

Listing 7: Object structure of request body for GetInverterRealtimeData request (Scope "Device")

```
object {

# Collection of named value-unit pairs according to selected DataCollection.
# Members of Data object are named according to the value they represent (e.g. "PAC").
  object {

    # Value-Unit pair.
    object {

      # Unscaled value.
      number Value;

      # Base unit of the value, never contains any prefixes.
      string Unit;

    } __VALUE_NAME__;

  }* Data;
};
```

### 3.1.5 Example of request body (Scope "Device")

Listing 8: Example of request body for GetInverterRealtimeData request (Scope "Device")

```
// GetInverterRealtimeData.cgi?Scope=Device&DeviceIndex=0&DataCollection=CommonInverterData

{
    "Data" : {
        "DAY_ENERGY" : {
                "Value" : 8000,
                "Unit" : "Wh"
        },
        "FAC" : {
                "Value" : 50,
                "Unit" : "Hz"
        },
        "IAC" : {
                "Value" : 14.54,
                "Unit" : "A"
        },
        "IDC" : {
                "Value" : 8.2,
                "Unit" : "A"
        },
        "PAC" : {
                "Value" : 3373,
```

```
                "Unit" : "W"
        },
        "SAC" : {
                "Value" : 3413,
                "Unit" : "VA"
        },
        "TOTAL_ENERGY" : {
                "Value" : 45000,
                "Unit" : "Wh"
        },
        "UAC" : {
                "Value" : 232,
                "Unit" : "V"
        },
        "UDC" : {
                "Value" : 426,
                "Unit" : "V"
        },
        "YEAR_ENERGY" : {
                "Value" : 44000,
                "Unit" : "Wh"
        },
        "DeviceStatus" : {
                "StatusCode" : 7,
                "MgmtTimerRemainingTime" : -1,
                "ErrorCode" : 0,
                "LEDColor" : 2,
                "LEDState" : 0,
                "StateToReset" : false
        }
    }
}
```


### 3.1.6 Object structure of request body (Scope "System")

Listing 9: Object structure of request body for GetInverterRealtimeData request (Scope "System")

```
object {

# Collection of named object(s) containing values per device and metadata.
# Members of Data object are named according to the value they represent (e.g. "PAC").
  object {

    # Value-Unit pair.
    object {

      # Base unit of the value, never contains any prefixes.
      string Unit;

      # Unscaled values per device.
      # Property name is the DeviceIndex to which the value belongs.
      object {

        number 1;  # value from device with index 1.
        number 2;  # value from device with index 2.
        # .. and so on.

      }* Values;

    } __VALUE_NAME__;

  }* Data;

};
```

### 3.1.7 Example of request body (Scope "System")

Listing 10: Example of request body for GetInverterRealtimeData request (Scope "System")

```
// GetInverterRealtimeData.cgi?Scope=System

{
    "Data" : {
        "PAC" : {
            "Unit" : "W",
            "Values" : {
                "0" : 4330,
                "1" : 3721
            }
        },
        "DAY_ENERGY" : {
            "Unit" : "Wh",
            "Values" : {
                "0" : 6366,
                "1" : 6000
            }
        },
        "YEAR_ENERGY" : {
            "Unit" : "Wh",
            "Values" : {
                "0" : 1041607,
                "1" : 42000
            }
        },
        "TOTAL_ENERGY" : {
            "Unit" : "Wh",
            "Values" : {
                "0" : 3551131,
                "1" : 43000
            }
        }
    }
}
```

## 3.2 *GetSensorRealtimeData* request

This request provides data for all channels of a single Fronius Sensor Card.
Inactive channels and channels with damaged sensors are not included in the response.

### 3.2.1 URL for HTTP requests

`/solar_api/GetSensorRealtimeData.cgi`

### 3.2.2 Parameters

| Parameter | Type | Range/Values/Pattern | Description |
|---|---|---|---|
| DeviceIndex | Integer | 0 . . . 9 | Which card to query. |
| DataCollection | String | "NowSensorData"<br>"MinMaxSensorData" | Selects the collection of data that should be queried from the device.<br>See 3.2.3 for details. |

### 3.2.3 Data Collections

**NowSensorData** The presently measured values of every active channel.

**MinMaxSensorData** The minimum and maximum values for every time period (day, month, year, total) of every channel.
Some channels do not have a minimum value because it would always be zero. For these channels, the minimum value is not included.

### 3.2.4 Object structure of request body (DataCollection "NowSensorData")

Listing 11: Object structure of request body for GetSensorRealtimeData request (DataCollection "NowSensorData")

```
object {

  # Collection of named object(s) containing values per channel and metadata.
  # Members of Data object are named according to the channel index they represent (e.g.
      "0").
  object {

    # Value-Unit pair.
    object {

      # Value for the channel.
      number Value;

      # Base unit of the value, never contains any prefixes.
      string Unit;

    } __CHANNEL_INDEX__;

  }* Data;

};
```

### 3.2.5 Example of request body (DataCollection "NowSensorData")

Listing 12: Example of request body for GetSensorRealtimeData request (DataCollection "NowSensorData")

```
// GetSensorRealtimeData.cgi?Scope=Device&DeviceIndex=1&DataCollection=NowSensorData

{
    "Data" : {
        "0" : {
            "Value" : -9,
            "Unit" : "\u00B0C"
        },
        "1" : {
            "Value" : 24,
            "Unit" : "\u00B0C"
        },
        "2" : {
            "Value" : 589,
            "Unit" : "W\/m\u00B2"
        },
        "4" : {
            "Value" : 0,
            "Unit" : "kWh\/m\u00B2"
        }
    }
}
```

### 3.2.6 Object structure of request body (DataCollection "MinMaxSensorData")

Listing 13: Object structure of request body for GetSensorRealtimeData request (DataCollection "MinMaxSensorData")

```
object {

    # Collection of named object(s) containing min/max values per channel and metadata.
    # Members of Data object are named according to the channel index they represent (e.g.
        "0").
```

```
    object {

        # Object representing one channel.
        object {

            # Whether this channel is currently active.
            boolean SensorActive;

            # Object representing min/max values of current day.
            object {

                # Maximum value with unit.
                object Max {
                    number Value;
                    string Unit;
                };

                # Minimum value with unit.
                # This object is only present in temperature channels (channel# 0 and 1)
                # as other channels do not have minimum values.
                object Min {
                    number Value;
                    string Unit;
                };
            } Day;

            # Object representing min/max values of current month.
            object {
                object Max {
                    number Value;
                    string Unit;
                };
                object Min {
                    number Value;
                    string Unit;
                };
            } Month;

            # Object representing min/max values of current year.
            object {
                object Max {
                    number Value;
                    string Unit;
                };
                object Min {
                    number Value;
                    string Unit;
                };
            } Year;

            # Object representing total min/max values.
            object {
                object Max {
                    number Value;
                    string Unit;
                };
                object Min {
                    number Value;
                    string Unit;
                };
            } Total;

        } __CHANNEL_INDEX__;

    }* Data;

};
```

### 3.2.7 Example of request body (DataCollection "MinMaxSensorData")

Listing 14: Example of request body for GetSensorRealtimeData request (DataCollection "MinMaxSensorData")

```
// GetSensorRealtimeData.cgi?Scope=Device&DeviceIndex=1&DataCollection=MinMaxSensorData

{
    "Data" : {
        "0" : {
            "SensorActive" : true,
            "Day" : {
                "Max" : {
                    "Value" : 7,
                    "Unit" : "\u00B0C"
                },
                "Min" : {
                    "Value" : -1,
                    "Unit" : "\u00B0C"
                }
            },
            "Month" : {
                "Max" : {
                    "Value" : 15,
                    "Unit" : "\u00B0C"
                },
                "Min" : {
                    "Value" : -9,
                    "Unit" : "\u00B0C"
                }
            },
            "Year" : {
                "Max" : {
                    "Value" : 35,
                    "Unit" : "\u00B0C"
                },
                "Min" : {
                    "Value" : -12,
                    "Unit" : "\u00B0C"
                }
            },
            "Total" : {
                "Max" : {
                    "Value" : 37,
                    "Unit" : "\u00B0C"
                },
                "Min" : {
                    "Value" : -15,
                    "Unit" : "\u00B0C"
                }
            }
        },
        "1" : {
            "SensorActive" : true,
            "Day" : {
                "Max" : {
                    "Value" : 7,
                    "Unit" : "\u00B0C"
                },
                "Min" : {
                    "Value" : -2,
                    "Unit" : "\u00B0C"
                }
            },
            "Month" : {
                "Max" : {
                    "Value" : 9,
                    "Unit" : "\u00B0C"
```

```
            },
            "Min" : {
                "Value" : -8,
                "Unit" : "\u00B0C"
            }
        },
        "Year" : {
            "Max" : {
                "Value" : 27,
                "Unit" : "\u00B0C"
            },
            "Min" : {
                "Value" : -11,
                "Unit" : "\u00B0C"
            }
        },
        "Total" : {
            "Max" : {
                "Value" : 31,
                "Unit" : "\u00B0C"
            },
            "Min" : {
                "Value" : -14,
                "Unit" : "\u00B0C"
            }
        }
    },
    "2" : {
        "SensorActive" : false,
        "Day" : {
            "Max" : {
                "Value" : 593,
                "Unit" : "W\/m\u00B2"
            }
        },
        "Month" : {
            "Max" : {
                "Value" : 785,
                "Unit" : "W\/m\u00B2"
            }
        },
        "Year" : {
            "Max" : {
                "Value" : 923,
                "Unit" : "W\/m\u00B2"
            }
        },
        "Total" : {
            "Max" : {
                "Value" : 932,
                "Unit" : "W\/m\u00B2"
            }
        }
    },
    "3" : {
        "SensorActive" : true,
        "Day" : {
            "Max" : {
                "Value" : 5,
                "Unit" : "km\/h"
            }
        },
        "Month" : {
            "Max" : {
                "Value" : 22,
                "Unit" : "km\/h"
            }
```

```
            },
            "Year" : {
                "Max" : {
                    "Value" : 54,
                    "Unit" : "km\/h"
                }
            },
            "Total" : {
                "Max" : {
                    "Value" : 56,
                    "Unit" : "km\/h"
                }
            }
        },
        "4" : {
            "SensorActive" : false,
            "Day" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "Wh"
                }
            },
            "Month" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "Wh"
                }
            },
            "Year" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "Wh"
                }
            },
            "Total" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "Wh"
                }
            }
        },
        "5" : {
            "SensorActive" : false,
            "Day" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "mbar"
                }
            },
            "Month" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "mbar"
                }
            },
            "Year" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "mbar"
                }
            },
            "Total" : {
                "Max" : {
                    "Value" : 0,
                    "Unit" : "mbar"
                }
            }
```

```
        }
    }
}
```

## 3.3 *GetStringRealtimeData* request

### 3.3.1 URL for HTTP requests

`/solar_api/GetStringRealtimeData.cgi`

### 3.3.2 Parameters

| Parameter | Type | Range/Values/Pattern | Description |
|---|---|---|---|
| Scope | String | "Device" | Query specific device |
| DeviceIndex | Integer | 0 ... 199 | Which device to query. |
| DataCollection | String | "NowStringControlData"<br>"LastErrorStringControlData"<br>"CurrentSumStringControlData" | Selects the collection of data that should be queried from the device. See 3.3.3 for details. |
| TimePeriod | String | "Day"<br>"Year"<br>"Total" | *Only needed for Collection "CurrentSumStringControlData"*<br>For which time period the current sums should be requested. |

### 3.3.3 Data Collections

**NowStringControlData** The presently measured currents of every channels.

**LastErrorStringControlData** Information about the last error which triggered a service message.

**CurrentSumStringControlData** Current sums of all channels for a selected time period (day, year or total).

### 3.3.4 Object structure of request body (DataCollection "NowStringControlData" and "CurrentSumStringControlData")

Listing 15: Object structure of request body for GetStringRealtimeData request (DataCollection "NowStringControlData" and "CurrentSumStringControlData")

```
object {

  # Collection of named object(s) containing values per channel and metadata.
  # Members of Data object are named according to the channel index they represent (e.g.
      "0").
  object {

    # Value-Unit pair.
    object {

      # Value for the channel.
      number Value;

      # Base unit of the value, never contains any prefixes.
      string Unit;

    } __CHANNEL_INDEX__;

  }* Data;

};
```

### 3.3.5 Example of request body (DataCollection "CurrentSumStringControlData")

Listing 16: Example of request body for GetStringRealtimeData request (DataCollection "CurrentSumStringControlData")

```
// GetStringRealtimeData.cgi?Scope=Device&DeviceIndex=1&DataCollection=
    CurrentSumStringControlData&TimePeriod=Day

{
    "1" : {
        "Value" : 59.57,
        "Unit" : "Ah"
    },
    "2" : {
        "Value" : 47.98,
        "Unit" : "Ah"
    },
    "3" : {
        "Value" : 47.6,
        "Unit" : "Ah"
    },
    "4" : {
        "Value" : 36.17,
        "Unit" : "Ah"
    },
    "5" : {
        "Value" : 47.91,
        "Unit" : "Ah"
    }
}
```

### 3.3.6 Object structure of request body (DataCollection "LastErrorStringControlData")

Listing 17: Object structure of request body for GetStringRealtimeData request (DataCollection "LastErrorStringControlData")

```
object {

    object {

        # Timestamp when the error was detected.
        string TimeOfError;

        # Average value of all channels
        # at the time the error was detected.
        object {

            number Value;

            # Base unit of the value, never contains any prefixes.
            string Unit;

        } StringAverage;

        # Contains information about every channel
        # at the time the error was detected.
        object {

            # Object representing one channel.
            object {

                # Deviation from string average.
                object {

                    number Value;
```

```
                    # Base unit of the value, never contains any prefixes.
                    string Unit;

                } Deviation;

                # Current sum
                object {

                    number Value;

                    # Base unit of the value, never contains any prefixes.
                    string Unit;

                } Sum;

            } __CHANNEL_INDEX__;

        }* Channels;

    } Data;
}
```

### 3.3.7 Example of request body (DataCollection "LastErrorStringControlData")

Listing 18: Example of request body for GetStringRealtimeData request (DataCollection "LastErrorStringControlData")

```
// GetStringRealtimeData.cgi?Scope=Device&DeviceIndex=1&DataCollection=
    LastErrorStringControlData
{
    "TimeOfError" : "2012-03-02T23:50:00+01:00",
    "StringAverage" : {
        "Value" : 46.22,
        "Unit" : "Ah"
    },
    "Channels" : {
        "1" : {
            "Deviation" : {
                "Value" : 24.7,
                "Unit" : "%"
            },
            "Sum" : {
                "Value" : 57.66,
                "Unit" : "Ah"
            }
        },
        "2" : {
            "Deviation" : {
                "Value" : 0.3,
                "Unit" : "%"
            },
            "Sum" : {
                "Value" : 46.4,
                "Unit" : "Ah"
            }
        },
        "3" : {
            "Deviation" : {
                "Value" : -0.7,
                "Unit" : "%"
            },
            "Sum" : {
                "Value" : 45.9,
                "Unit" : "Ah"
            }
        },
```

```
        "4" : {
            "Deviation" : {
                "Value" : -24.5,
                "Unit" : "%"
            },
            "Sum" : {
                "Value" : 34.87,
                "Unit" : "Ah"
            }
        },
        "5" : {
            "Deviation" : {
                "Value" : 0.1,
                "Unit" : "%"
            },
            "Sum" : {
                "Value" : 46.29,
                "Unit" : "Ah"
            }
        }
    }
}
```

## 3.4 *GetLoggerInfo* request

This request provides information about the logging device which provides this API.

### 3.4.1 URL for HTTP requests

`/solar_api/GetLoggerInfo.cgi`

### 3.4.2 Object structure of request body

Listing 19: Object structure of request body for GetLoggerInfo request

```
object {

  object {

    # Unique ID of the logging device.
    string UniqueID;

    # Hardware version of the logging device.
    string HWVersion;

    # Software version of the logging device.
    string SWVersion;

    # Name of city/country which the user
    # selected as time zone.
    string TimezoneLocation/[a-zA-Z]+|/;

    # Name of the selected time zone.
    # May be empty if information not available.
    string TimezoneName/[a-zA-Z]+|/;

    # UTC offset in seconds east of UTC,
    # including adjustments for daylight saving.
    integer UTCOffset;

    # Default language set on the logging device
    # as a two letter abbreviation (e.g. "en").
    string DefaultLanguage;

    # The cash factor set on the logging device,
```

```
    # NOT the factor set on the inverters.
    number CashFactor;

    # Currency of cash factor set on the logging device,
    # NOT the currency set on the inverters.
    string CashCurrency;

    # The CO2 factor set on the logging device,
    # NOT the factor set on the inverters.
    number CO2Factor;

    # Unit of CO2 factor set on the logging device,
    # NOT the unit set on the inverters.
    string CO2Unit;

  } LoggerInfo;

};
```

### 3.4.3 Example of request body

Listing 20: Example of request body for GetLoggerInfo request

```
// GetLoggerInfo.cgi
{
    "LoggerInfo": {
        "UniqueID": "240.2",
        "HWVersion": "1.4A",
        "SWVersion": "2.0.2-7",
        "TimezoneLocation": "Amsterdam",
        "TimezoneName": "CEST",
        "UTCOffset": 7200,
        "DefaultLanguage" : "de",
        "CashFactor" : 0.47,
        "CashCurrency" : "EUR",
        "CO2Factor" : 0.53,
        "CO2Unit" : "kg"
    }
}
```

## 3.5 *GetInverterInfo* request

This request provides information about all inverters that are currently being monitored by the logging device. So this means that inverters which are currently not online are also reported by this request, provided these inverters have been seen by the logging device within the last 24 hours.

If information about devices currently online is needed, the *GetActiveDeviceInfo* request should be used. This request also provides information about device classes other than inverters.

### 3.5.1 URL for HTTP requests

`/solar_api/GetInverterInfo.cgi`

### 3.5.2 Object structure of request body

Listing 21: Object structure of request body for GetInverterInfo request

```
object {

  # Collection of objects with infos about one inverter,
  # mapped by inverter index.
  object {
```

```
    # Info about a single inverter.
    # Name of object is the inverter index.
    object {

      # Device type of the inverter.
      integer DT;

      # PV power connected to this inverter (in watts).
      # If none defined, default power for this DT is used.
      integer PVPower;

      # Unique ID of the inverter (e.g. serial number).
      string UniqueID;

      # Error code that is currently present on inverter.
      # A value of -1 means that there is no valid error code.
      number ErrorCode;

      # Status code reflecting the operational state of the inverter.
      number StatusCode;

    } __INVERTER_INDEX__;

  }* Data;

};
```

### 3.5.3 Example of request body

Listing 22: Example of request body for GetInverterInfo request

```
// GetInverterInfo.cgi

{
  "Data" : {
      "1" : {
          "DT" : 192,
          "PVPower": 5000,
          "UniqueID": "123456",
          "ErrorCode" : 0,
          "StatusCode" : 7
      },
      "2" : {
          "DT" : 192,
          "PVPower": 5000,
          "UniqueID": "234567",
          "ErrorCode" : 0,
          "StatusCode" : 7
      }
  }

}
```

### 3.5.4 Meaning of numerical status codes

The `StatusCode` Field is only reported as numerical value. The meaning of the numbers is shown in the table below.

| Value | Description |
|---|---|
| 0 - 6 | Startup |
| 7 | Running |
| 8 | Standby |
| 9 | Boot loading |
| 10 | Error |

## 3.6 *GetActiveDeviceInfo* request

This request provides information about which devices are currently online.

### 3.6.1 URL for HTTP requests

`/solar_api/GetActiveDeviceInfo.cgi`

### 3.6.2 Parameters

| Parameter | Type | Range/Values/Pattern | Description |
|---|---|---|---|
| DeviceClass | String | "Inverter"<br>"SensorCard"<br>"StringControl" | Which kind of device class to search for active devices. |

### 3.6.3 Object structure of request body

Listing 23: Object structure of request body for GetActiveDeviceInfo request

```
object {

  # Collection of objects with infos about one inverter,
  # mapped by inverter index.
  object {

    # Info about a single device.
    # Name of object is the device index.
    object {

      # Device type of the device.
      integer DT;

    } __DEVICE_INDEX__;

  }* Data;

};
```

### 3.6.4 Example of request body

Listing 24: Example of request body for GetActiveDeviceInfo request

```
// GetActiveDeviceInfo.cgi?DeviceClass=Inverter

{
  "Data" : {
      "1" : {
          "DT" : 192
      },
      "2" : {
          "DT" : 192
      }
  }

}
```

# 4 Frequently asked questions

1. **Which data can I get?**
   Currently only realtime data from inverters, Fronius Sensor Cards and Fronius String Controls. Also some information about the logging device itself is available.
   Please refer to the API specs for further details.

2. **Can multiple clients send requests to the API at the same time?**
   Yes, but the requests may take longer to complete.

3. **Can I use this API at the same time as other services of the Fronius Datalogger Web?**
   Yes. The datalogging, Solar.access/Solar.web connection, Webinterface, this API or any other service can be used independently from the others.

4. **Can the API calls be password protected?**
   No. The API is always accessible without authentication, regardless of the user or admin password set on the Webinterface.

5. **The API reports more inverters than I have, why is that?**
   This may be the case when the inverter number of an inverter is changed while the Fronius Datalogger Web is running. The logger then detects a new device but keeps the same device with the previous inverter number in the system for 24 hours. This is due to the fact that the datalogger is caching the devices for a certain time even if they are not present on the bus (e.g. to be able to display energy values during the night when the inverters are offline).
   Those ghost devices will disappear 24 hours after the have been last seen by the datalogger. Alternatively, a reboot of the datalogger also clears the device cache and repopulates it with the currently present devices.

# Fronius Worldwide - www.fronius.com/addresses


(A) **Fronius International GmbH**
4600 Wels, Froniusplatz 1, Austria
E-Mail: pv@fronius.com
http://www.fronius.com

(USA) **Fronius USA LLC** Solar Electronics Division
6797 Fronius Drive, Portage, IN 46368
E-Mail: pv-us@fronius.com
http://www.fronius-usa.com


**Under http://www.fronius.com/addresses you will find all addresses of our sales branches and partner firms!**

ud_fr_se_so_00913 012013