HydroMet



Supplement to Edge Device User Manuals

Communication Protocols and File Formats

KISTERS
Empowering decisions of tomorrow

11-2026 – 01-00-001

Table of Contents

1 Introduction 3

2 KISTERS Edge Devices 4

3 File Transfer 5

3.1 SFTP File Transfer 5

3.1.1 Device Configuration 5

3.2 HTTPS File Transfer 5

4 File Formats – Structural Description 6

4.1 MIS 6

4.1.1 File structure 6

4.1.2 Parser implementation guidance 7

4.2 ZRXP 2.3 7

4.2.1 File structure 8

4.2.2 Main elements and attributes 8

4.2.3 Parser implementation guidance 9

4.3 HailSens IoT JSON 9

4.3.1 File structure 10

4.3.2 Main elements and attributes 10

4.3.3 Parser implementation guidance 12

4.3.4 Compatibility note 12

4.4 HailSens IoT JSON via REST API 13

4.4.1 File structure 13

4.4.2 Main elements and attributes 13

4.4.3 REST API 13

4.5 OTTML data format 13

4.5.1 File structure 14

4.5.2 Main elements and attributes 15

4.5.3 Channel organisation 15

4.5.4 Optional metadata and extensions 16

4.5.5 Parser implementation guidance 16

© 2026 | KISTERS

1 Introduction

This document describes the structure of the data file formats supported by KISTERS Edge devices. It is primarily intended to provide software developers and system integrators with the information required to implement reliable parsers and import interfaces for data generated by KISTERS Edge devices.

2 KISTERS Edge Devices

KISTERS Edge devices are based on the common KISTERS KIPTEC technology platform, providing a shared foundation for data acquisition, processing, communication and IT security. While individual Edge devices support different measurements and applications, they use common mechanisms for storing and transferring measurement and diagnostic data.

To facilitate integration with existing and new data infrastructures, KISTERS Edge devices support established file formats such as MIS and OTTML. The available measurement channels and their content depend on the individual device and configuration, while the fundamental file structures described in this document are common across the KISTERS Edge family.

3 File Transfer

The following chapters describe the available file transfer protocols.

- SFTP File Transfer 5
- HTTPS File Transfer 5

3.1 SFTP File Transfer

Measurement data can be transferred automatically from KISTERS Edge devices to a configured server using SFTP (SSH File Transfer Protocol). SFTP provides encrypted and authenticated file transfer and is the recommended option for modern installations.

Acronym	Meaning	Supported by Edge Device	Description
SFTP	SSH File Transfer Protocol	✓	runs over SSH. It does not use TLS.
FTPS	FTP Secure / FTP over TLS	✗	conventional FTP secured with TLS.
FTP	File Transfer Protocol	✗	unencrypted legacy protocol

Where compatibility with existing infrastructure is required, legacy FTP and SFTP can coexist on the same server, provided that the server is configured to support both services. This allows existing FTP-based workflows to be maintained while new or migrated devices use secure SFTP transfer.

3.1.1 Device Configuration

Use KISTERS' HyComm software to set up an Edge device for SFTP file transfer.

Specify the following connection parameters:

- Host: The hostname or IP address of the SFTP server, e.g., sftp.your-domain.de.
- Port: The SFTP server port. If left blank, HyComm automatically uses the default SFTP port 22.
- Credentials: Select one of the following authentication methods:
  - Username and password, or
  - Username and SSH key. The device provides the public key, which must be added to the SFTP server.

3.2 HTTPS File Transfer

Secure communication is an integral part of the KISTERS KIPTEC platform. In addition to SFTP, KIPTEC-based communication interfaces use modern encrypted communication mechanisms, including TLS 1.3 where TLS-based protocols are used.

4 File Formats – Structural Description

The following chapters describe the available file formats.

- MIS 6
- ZRXP 2.3 7
- HailSens IoT JSON 9
- HailSens IoT JSON via REST API 13
- OTTML data format 13

4.1 MIS

Topic	Mode	Edge Release	HyComm Release
Transport Protocol	SFTP	2.7.10	
File Format	MIS	2.7.10	

The device supports the export of measurement data in .mis file format.

The .mis format is a structured, text-based data file format commonly used for the exchange of environ-mental and meteorological measurement data between sensors, data loggers, and backend systems.

Typical characteristics include:

- ASCII text-based structure, enabling human readability and easy processing.
- A header followed by one or more measurement records.
- Delimited fields containing measured parameters, timestamps, and metadata: Individual records are separated by line breaks. Fields within a record are separated by semicolons (;).
- Record-oriented organization, where each record represents a measurement set or timestamped observation.
- Inclusion of device identifiers, timestamps, and sensor values in a predefined order.

4.1.1 File structure

A .mis file consists of one or more sensor blocks. Each block contains the measurement records for one measurement channel or diagnostic parameter. Applications importing .mis files should therefore be capable of handling optional fields and format variations.

For the.mis variant generated by KISTERS Edge devices, a block has the following structure:

```
<STATION>station_id</STATION><SENSOR>sensor_id</SENSOR><DATEFORMAT>YYYYMMDD</DATEFORMAT>
date;time;value
date;time;value
...
```

The individual elements have the following meaning:

Element	Description
`STATION`	Identifier of the station/device to which the following records belong.
`SENSOR`	Identifier of the measurement channel or parameter. The identifier may be descriptive text and must not be assumed to be numeric.
`DATEFORMAT`	Defines the date representation used in the following records. HyQuant uses `YYYYMMDD`.
`date`	Measurement date according to `DATEFORMAT`, e.g. `20260706`.
`time`	Measurement time in `HHMMSS` format, e.g. `143500`.

Element	Description
`value`	Measured or diagnostic value represented as text. Values may, for example, be integers, decimal numbers or negative numbers.

Fields in a measurement record are separated by semicolons (;). Each measurement record occupies one line. A new header line starts a new sensor block.

Different sensor blocks do not necessarily contain the same number of records or the same timestamps. Parameters may be recorded at different intervals; for example, measurement values may occur periodically while diagnostic values are recorded less frequently. A parser must therefore process each sensor block independently and must not assume that records in different blocks can be matched by line number.

4.1.2 Parser implementation guidance

For robust parsing, applications should:

- detect the start of a block from the header tags rather than from a fixed line position;
- treat the contents of `STATION` and `SENSOR` as identifiers rather than relying on a predefined list or numeric format;
- use the `DATEFORMAT` information when interpreting the date;
- split measurement records into exactly three logical fields: date, time, and value;
- allow different numbers and timestamps of records between sensor blocks;
- tolerate empty lines between blocks;

avoid deriving engineering units or the physical meaning of a value from the `.mis` structure itself, as these are not included in the measurement record.

The following simplified logical structure may be used as a basis for parser implementation:

```
MIS file
├─ Sensor block
│  ├─ Header
│  │  ├─ STATION
│  │  ├─ SENSOR
│  │  └─ DATEFORMAT
│  └─ 1..n measurement records
│  └─ date ; time ; value
├─ Sensor block
│  └─ ...
└─ ...
```

Implementation note: `.mis` should be treated as a family of related text formats rather than as one strictly standardised schema. Other devices or software generations may use different header contents, identifiers or record variants. Import software intended to support .mis files from multiple sources should therefore identify and parse the actual structure encountered instead of assuming that all `.mis` files are identical.

4.2 ZRXP 2.3

Topic	Mode	Edge Release	HyComm Release
Transport Protocol	SFTP	2.7.10	
File Format	ZRXP 2.3	2.7.10	

ZRXP (Zentraler Rohdaten-Exchange-Prozess) is a structured, text-based data exchange format originally developed by KISTERS for transferring hydrological and meteorological time-series data. Through widespread adoption by manufacturers of measurement equipment, data loggers and third-party software providers, ZRXP has become an established de facto industry standard for exchanging environmental monitoring data.

KISTERS Edge devices support ZRXP version 2.3, providing compatibility with existing KISTERS and 3rd-party data acquisition and data management systems.

Typical characteristics include:

- Latin-1 text-based encoding, allowing human readability and straightforward parsing
- Time series-oriented structure, representing sequences of timestamped measurement values
- Delimited records, where each record corresponds to a single observation (timestamp + value)
- Inclusion of metadata elements, such as station identifiers, parameter codes, units, and quality indicators
- Support for header sections defining context, data source, and configuration parameters

The format is designed for efficient exchange of measurement time series between data loggers, sensors, and backend systems, particularly in hydrological and meteorological applications.

4.2.1 File structure

ZRXP is a line-oriented text format using ISO-8859-1 (Latin-1) encoding. A file consists of one or more time-series segments, each containing a header followed by measurement records.

The structure generated by KISTERS Edge devices is:

```
ZRXP file
├── Time-series segment
│   ├── Header (station, channel, unit, layout)
│   └── 1..n measurement records
├── Time-series segment
│   ├── Header
│   └── 1..n measurement records
└── ...
```

Header lines begin with `#` and contain keyword/value pairs separated by `|*|` or `;*;`. Comments begin with `##`. The order of header keywords is not fixed.

The LAYOUT keyword defines the fields and their order in the measurement records. KISTERS Edge devices use:

```
#LAYOUT(timestamp,value,status,remark)|*|
```

A typical measurement record is:

```
20261022143000 52.04672 200
```

The fields are separated by whitespace. The final `remark` field may be omitted when empty.

4.2.2 Main elements and attributes

The following header keywords are used in the supplied KISTERS Edge implementation:

Keyword	Description
`ZRXPVERSION`	ZRXP format version
`ZRXPCREATOR`	Software generating the file
`SANR, SNAME`	Station identifier and name
`CDASA, CDASANAME`	Data logger identifier and name
`CCHANNELNO, CNAME`	Channel number and parameter name
`REXCHANGE`	Time-series exchange identifier

Keyword	Description
CUNIT	Engineering unit
TZ	Time zone applicable to timestamps
RINVAL	Marker for missing or invalid values
LAYOUT	Definition of measurement record fields

The measurement records contain:

Field	Description
timestamp	Measurement timestamp, typically YYYYMMDDhhmmss
value	Numeric measurement or diagnostic value
status	Encoded quality/status information (0–255)
remark	Optional text associated with the record

Each segment represents one measurement or diagnostic channel. Different channels may contain different numbers of records and sampling intervals.

4.2.3 Parser implementation guidance

Applications processing ZRXP files generated by KISTERS Edge devices should:

- Identify segments from their headers rather than fixed line positions.
- Interpret header keywords independently of their order.
- Use LAYOUT to identify measurement record fields.
- Interpret timestamps using the time zone specified by TZ.
- Process channels independently, without assuming identical timestamps or recording intervals.
- Recognise missing values according to RINVAL.
- Allow omitted or empty optional remarks.
- Treat channel names and engineering units as configurable metadata.

Implementation note: The structure described here reflects the ZRXP 2.3 implementation used by KISTERS Edge devices. For the complete format specification, including additional keywords, alternative record layouts, status information and time-series identification rules, refer to KISTERS – Data Exchange Format ZRXP, User Manual, Version 2.3 (2021).

4.3 HailSens IoT JSON

Topic	Mode	Edge Release	HyComm Release
Transport Protocol	SFTP	2.7.10	
File Format	HailSens IoT JSON Legacy format	2.7.10	

HailSens Edge devices support a legacy JSON (JavaScript Object Notation) data format for compatibility with existing data-processing systems originally developed for the predecessor HailSens IoT. JSON is a structured, text-based format that represents data using objects, arrays and key/value pairs.

For file-based transfer, the JSON files are sent to the configured server via SFTP. The structure described below defines the file content independently of the transfer method.

4.3.1 File structure

Each JSON file contains one top-level object consisting of a device identifier and three arrays:

```
{
  "device_id": "DEVICE_ID",
  "logs": [],
  "sensors": [
    {
      "id": 5,
      "key": "SENSORIP",
      "timestamp": 1783599975000,
      "value": "NOTHINGFOUND"
    }
  ],
  "values": [
    {
      "id": 5,
      "key": "BV",
      "sensor": "DEVICE_ID",
      "timestamp": 1783599600000,
      "value": "11.841"
    }
  ]
}
```

4.3.2 Main elements and attributes

The three arrays are sensors, values and logs. Each array may contain zero, one or multiple JSON objects. The actual number and combination of entries depend on the data available at the time the file is generated. This variable structure is an inherent characteristic of the legacy format. The original implementation deliberately omitted redundant information to reduce the transmitted data volume.

Element	JSON type	Description
device_id	String	Unique identifier of the originating device.
sensors	Array	Contains information identifying or describing the connected sensor.
values	Array	Contains measurement, status and diagnostic values.
logs	Array	Contains log messages, if available.

4.3.2.1 sensors array

Each entry in the sensors array is a JSON object with the following structure:

Key	JSON type	Description
`key`	String	Sensor information identifier. SENSORIP is used by the legacy implementation.

Key	JSON type	Description
`timestamp`	Number	Timestamp in milliseconds since 1970-01-01 00:00:00 UTC (Unix epoch).
`id`	Number	Legacy numeric identifier.
`value`	String	Sensor information associated with key.

The Edge files may use `SENSORIP` and may return `NOTHINGFOUND` rather than an IP address. Applications should therefore treat this field as descriptive legacy information and should not require it to contain a syntactically valid IP address.

4.3.2.2 values array

The values array contains the actual measurement, status and diagnostic information. Each entry represents one timestamped value.

Key	JSON type	Description
`key`	String	Identifier defining the meaning of the value.
`timestamp`	Number	Timestamp of the value in milliseconds since the Unix epoch.
`sensor`	String	Identifier of the sensor/device associated with the value.
`id`	Number	Legacy numeric identifier.
`value`	String or Number	Measurement, status or diagnostic value. The JSON datatype may vary depending on the parameter and generating software version.

4.3.2.3 key

The key determines how the associated value is to be interpreted. The legacy format defines the following identifiers:

Key	Description	Unit	Typical content
`BV`	Supply voltage	V	Status / heartbeat
`BT`	Internal sensor temperature	°C	Status / heartbeat
`C1`	Internal sensor status	–	Status / heartbeat
`V1`	Software revision - SensorBoardReader	–	Status / heartbeat
`V2`	Software revision - RestWriter	–	Status / heartbeat

Key	Description	Unit	Typical content
V3	Software revision – GateKeeper	–	Status / heartbeat
CF	Calibration information	–	Measurement data
D1	Hailstone diameter calculated from E1	mm	Measurement data
E1	Kinetic energy of impact	J	Measurement data
T1	Sensor raw value of impact	–	Measurement data
EV	Start of a new event (1)	–	Measurement data

The distinction between heartbeat/status information and event measurement data originates from the legacy implementation. Both use the same `values` array and object structure.

4.3.2.4 logs array

The optional logs array contains diagnostic or log messages:

Key	JSON type	Description
id	Number	Log entry identifier.
message	String	Log message text.
timestamp	Number	Timestamp of the log entry.

Caution: The array may be empty.

4.3.3 Parser implementation guidance

Applications processing the legacy JSON format should:

- use a standards-compliant JSON parser rather than relying on element order or textual layout;
- use `device_id` and sensor as identifiers rather than assuming a particular identifier format;
- identify the meaning of entries in `values` from their `key`;
- interpret `timestamp` as milliseconds since the Unix epoch;
- allow `sensors`, `values` and `logs` arrays to contain zero or multiple entries;
- not assume that every supported key is present in every file;
- not assume a fixed order of keys or array entries;
- accept `value` as either a JSON string or number and perform type conversion according to the associated `key`;
- tolerate additional keys or entries introduced by compatible implementations or later software versions.

Applications should not use the presence or absence of a particular status value to determine whether a JSON document is structurally valid.

4.3.4 Compatibility note

This JSON structure originates from an earlier KISTERS device and software generation and is retained by KISTERS Edge devices for compatibility with existing installations and data-processing systems. Applications should therefore be tolerant of optional or omitted entries and minor variations between implementations.

4.4 HailSens IoT JSON via REST API

Topic	Mode	Edge Release	HyComm Release
Transport Protocol	HTTPS	2.7.10	
File Format	HailSens IoT JSON Legacy format	2.7.10	

HailSens Edge devices support a legacy JSON (JavaScript Object Notation) data format for compatibility with existing data-processing systems originally developed for the predecessor HailSens IoT. JSON is a structured, text-based format that represents data using objects, arrays and key/value pairs.

For file-based transfer, the JSON files are sent to the configured server via SFTP. The structure described below defines the file content independently of the transfer method.

4.4.1 File structure

See ch. "HailSens IoT JSON" > File structure 10.

4.4.2 Main elements and attributes

See ch. "HailSens IoT JSON" > Main elements and attributes 10.

4.4.3 REST API

4.4.3.1 RESTful API – Return Codes

- Code of 201: confirms successful reception of the JSON datagram.
- All other http return codes are treated as not successful:

```
{
"error": "Unauthorized"
}
```

4.5 OTTML data format

Topic	Mode	Edge Release	HyComm Release
Transport Protocol	HTTPS	2.7.10	
File Format	OTTML, XML variant	2.7.10	

This interface enables:

- Retrieval and transmission of measurement data in OTTML file format via HTTPS requests
- Integration with existing REST-based systems
- Secure communication based on mutual TLS 1.3 authentication

The device supports the export of measurement data in OTTML (OTT Markup Language) file format. The data is sent in the body of a POST request to the configured URL.

OTTML is an XML-based data exchange format developed by OTT HydroMet for the transfer of hydrological, meteorological, and environmental monitoring data. In addition to measurement values, OTTML can include extensive metadata describing the monitoring station, sensors, device status, communication parameters, and system configuration.

Typical characteristics include:

- XML-based structure, providing a standardized and self-describing data format
- Human-readable text format that can be viewed and processed using standard XML tools
- Hierarchical organization of stations, channels, measurements, and metadata
- Support for timestamped measurement values and associated quality information
- Inclusion of station identifiers, channel identifiers, units, and configuration parameters
- Transmission of device status information, such as firmware version, battery voltage, signal strength, location, and communication settings

OTTML is particularly suited for automated data exchange between field devices, telemetry systems, and environmental data management platforms.

4.5.1 File structure

OTTML is an XML-based format used for exchanging measurement data and associated metadata. KISTERS Edge devices use a hierarchical structure in which measurement channels and their time-series values are grouped under a station.

The basic structure is:

```
StationDataList
 └─ StationData
     ├─ StationInfo
     ├─ ChannelData
     │   └─ Values
     │       ├─ VT
     │       ├─ VT
     │       └─ ...
     ├─ ChannelData
     │   └─ Values
     │       └─ ...
     └─ ...
```

The XML document starts with the standard `XML` declaration. `StationDataList` is the root element. Each `StationData` element represents one station or device and contains station-level information followed by one or more `ChannelData` elements.

A simplified example is shown below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StationDataList>
    <StationData stationId="DEVICE_ID"
                 name="DEVICE_NAME"
                 timezone="+00:00">
        <StationInfo firmware="2.7.6" />

        <ChannelData channelId="0001"
                     name="Measurement
                     parameter" unit="m">
                <Values>
                        <VT t="2026-07-09T11:04:00">1.234</VT>
                        <VT t="2026-07-09T11:05:00">1.235</VT>
                </Values>
        </ChannelData>
    </StationData>
</StationDataList>
```

4.5.2 Main elements and attributes

Element/attribute	Descriptions
`StationDataList`	Root element containing the station data included in the file.
`StationData`	Contains data and metadata belonging to one station/device.
`stationId`	Identifier of the station/device.
`name`	Descriptive station/device name, if provided.
`timezone`	Time-zone offset applicable to the timestamps contained in the station data.
`StationInfo`	Contains optional station/device metadata. The available attributes may depend on the device and software version.
`firmware`	Firmware version reported by the device, if provided.
`ChannelData`	Contains data belonging to one measurement or diagnostic channel.
`channelId`	Identifier of the channel.
`name`	Descriptive channel name.
`unit`	Engineering unit of the channel. The attribute may be empty for dimensionless values or status information.
`Values`	Container for the time-series values of a channel.
`VT`	Individual timestamped value.
`t`	Timestamp associated with a VT value.

The content of a VT element represents the actual value: `<VT t="2026-07-09T11:04:00">1.234</VT>`

The timestamp uses the form: `YYYY-MM-DDThh:mm:ss`

The `timezone` attribute of `StationData` provides the corresponding time-zone offset. Applications should use both when deriving an absolute timestamp.

4.5.3 Channel organisation

Each measurement, calculated value, diagnostic parameter or quality value is represented as a separate `ChannelData` element. Channel identification and description are therefore part of the XML data rather than being inferred from the position of the channel within the file.

For example:

```
<ChannelData channelId="0001" name="Measurement parameter" unit="m">
```

Applications should use `channelId` as the channel identifier. name and unit provide descriptive metadata and should not be assumed to remain identical between different device types, configurations or software versions.

Each channel contains its own Values collection. Channels must therefore be processed independently. Applications must not assume that:

- all channels contain the same number of values;
- all channels use the same storage interval;
- corresponding values occur at the same position within different channels; or
- every timestamp occurs in every channel.

Where values from different channels need to be correlated, this should be done using their timestamps.

4.5.4 Optional metadata and extensions

OTTML supports additional metadata and elements beyond the minimum structure required to represent a time series. Depending on the generating device, configuration and software version, additional station information, channel information, events or value attributes may be present.

A value may, for example, contain additional attributes describing an acquisition error:

```
<VT t="2026-07-09T11:04:00"
    errorcode="5"
    errortext="Line Break">0</VT>
```

Import applications should therefore use an XML parser rather than processing OTTML as fixed-position text. Unknown optional elements or attributes should not cause an otherwise valid data file to be rejected.

4.5.5 Parser implementation guidance

For robust processing of OTTML files generated by KISTERS Edge devices:

- parse the document according to its XML hierarchy rather than relying on element positions or line numbers;
- identify stations using `StationData` and their `stationId`;
- identify channels using `ChannelData` and `channelId`;
- process each `ChannelData` element independently;
- obtain measurement values from the text content of `VT`;
- obtain the corresponding timestamp from the t attribute;
- take the `StationData` timezone attribute into account when interpreting timestamps;
- treat descriptive attributes such as name and unit as metadata rather than as stable identifiers;
- allow optional attributes and elements;
- tolerate additional XML elements introduced by other OTTML variants or later software versions;
- use normal XML entity decoding when reading attribute values and element contents.

Applications should not derive the physical meaning, permissible range or quality interpretation of a channel solely from the generic OTTML structure. Such information is device- and channel-specific and is described separately in the applicable device documentation.

Contact Data

Europe	KISTERS Europe	+49 2408 9385 0 hydromet.sales@kisters.eu www.kisters.eu
Australia	KISTERS Australia	+612 9601 2022 sales@kisters.com.au www.kisters.com.au
New Zealand	KISTERS New Zealand	+64 7 857 0810 sales@kisters.co.nz www.kisters.co.nz
Latin America	KISTERS Latin America	+57 350 575 4079 sales@kisters-latam.com www.kisters-latam.com
North America	KISTERS North America	+1 561 459 4876 kna@kisters.net www.kisters.net
Spain	KISTERS Ibérica	info@kisters.es www.kisters.es
France	KISTERS France	+33 1 4120 9200 info@kisters.fr www.kisters.fr
Canada	KISTERS Canada	kna@kisters.net www.kisters.net

© 2026 | KISTERS

www.kisters.eu
© 2026 | KISTERS
05/11/2026

KISTERS
Empowering decisions of tomorrow