All the technical data for the Tani historian including its api are documented here.

General, File formats, REST interface

Historian General Technical Data

If the historian is installed in OPC UA the configured variables will offer the history functionality.

Operation mode: File

The maximum memory used can be limited, on default the limit is using memory until 80% of the system memory. On startup of the historian or on all changes in the historian configuration the memory limits are checked.

Operation mode: Disc

The maximum disk usage is depending on the share or mount the history is written to. During the historian installation a base point will be configured.
- On Linux this is a directory. This directory can be a mount also.
- Under Windows this is a drive or network share and a directory.
Each configured history variable will use a created sub directory under the base point. The directory name is the variable name. If the multiple variabled need to be written on separate disc drives you can mount each variable directory to another drive.
The disc variables ars always in sub directories under the configured base point.

Each variables space limit can be configured separately. Thereby the global setting can be used, or the variable is using private settings. Please beware setting the security limits to small, this can damage your machine.
All default settings are save always.

Tani Historian File Description

Directory structure

The historian has a configurable data directory. All historian variables that use file storage write the data in this directory. It can be changed in <install directory>/Historian/bin/Historian.Settings, key "HistorianFilePath".

Below this directory, each historian variable has its own sub-directory (name is the variable name). Each directory contains these files:

- Var.ini: is a copy of the variable configuration. This allows reconstructing the variable settings and file formats, even if the variable itself has been deleted from the historian.
- Structs.ini: exists only for variables that have a structure type. It contains all required structure definitions.
- data_*.bin: are the data files.

Var.ini contents

The Var.ini file is an ini-File with one section. The section name is ['Var.<variable name>]'. In this section, the following keys can exist:

Active
: Boolean; yes enables the variable in the runtime system.

AggregateAvg
: Boolean; yes enables calculation of Average

AggregateCalcOffset
: Number; for DataType=struct, the offset of the element that is used for aggregate calculation

AggregateCalcDataType
: Enum; for DataType=struct, the data type of the element that is used for aggregate calculation; seeData Types

AggregateCount
: Boolean; yes enables calculation of Count

AggregateMax
: Boolean; yes enables calculation of Maximum

AggregateMin
: Boolean; yes enables calculation of Minimum

AggregateStddev
: Boolean; yes enables calculation of Standard Derivation

ArrayLength
: Number; number of array elements to store

DataType
: Enum; data type of value to store; seeData Types; for structures, a : and the structure name is appended

ElementLength
: Number; for DataType=string, the maximum string length; for DataType=struct, the data length of the structure

FileResolution
: Enum; The resolution for creating new data files; seeResolution

FileSave
: Boolean; yes to enable file save

MaxFileCount
: Number; the maximum number of live data value files to create

MaxMem
: Number; number of bytes to use for live data storage of memory variable

MinimumFreeDiskSpace
: Number; mimimal required free disk space; 0 uses globally defined value; values < 100 are percent; value >= 100 are bytes

OPCActive
: Boolean; yes to enable automatic data acquisation of OPC data

OPCConnName
: String; OPC Connection name to use

OPCGroupName
: String; OPC Connection group name to use

OPCVarName
: String; OPC Variable name to use

ReadResolution
: Enum; The resolution of OPC data acquisation; seeResolution

Structs.ini contents

The Structs.ini is an ini-file with multiple sections. Each section describes one data structure used in the data files. The section name is equal to the structure name, some characters with special meaning are replaced by <xx> where xx is the hex-code of the character. The sequence of section entries describes the sequence of

structure elements. Valid keys are:

Header
<Structure Name>,<Flags>
<Structure Name> - the structure name (repeated, for historical reasons)
<Flags> - hexadecimal encoded flag bits; internal use only; not required for decoding
Element
<Element Name>,<Data Type>,<Data Type Flags>,<Element Flags>,<Array Size>,<Original Type>
<Element Name> - the element name; some characters with special meaning are replaced by<xx> where xx is the hex-code of the character.
<Data Type> - data type of the structure element; seeData Types; for structures and enumerations, a : and the structure/enumeration name is appended; for strings, a : and the maximum string length is appended
<Data Type Flags> - hexadecimal encoded flag bits; internal use only; not required for decoding
<Element Flags> - hexadecimal encoded flag bits; valid values from the following list are OR'ed tokether: 0x0010 - PACKED - if set, no bit padding exists before this element
0x0020 - PADDING - if set, this element is a padding element and can be ignored or hidden by user application
0x0040 - VARIABLE_ARRAY_LENGTH - if set, the array length is given by another structure element instead of the ArrayLength field
0x0080 - VARIABLE_STRING_LENGTH - if set, the string length is dynamic, instead of a fixed string length with padding
0x0100 - OPTIONAL - if set, the element is optional; its presence is determined by another structure element
all other bits are reserved for internal use only
<Array Size> - the array length in elememnts <Original Type> - a protocol specific "original type" for imported structs; may be safely ignored
ElementDynamic
optional; <Length Element>,<Switch Element>,<Switch Value>; this line is present only if at least one of these settings are required
<Length Element> - used if VARIABLE_ARRAY_LENGTH is set; the name of a structure element that gives the array length of this element; some characters with special meaning are replaced by <xx> where xx is the hex-code of the character.
<Switch Element> - used if OPTIONAL is set; the name of a structure element that determines if this element is present or not; some characters with special meaning are replaced by <xx> where xx is the hex-code of the character.
<Switch Value> - used if OPTIONAL is set; if the <Switch Element> has this value, then this element is present, otherwise it is absent
ElementComment
optional; the element comment; some characters with special meaning are replaced by<xx> where xx is the hex-code of the character.

Data file contents

The data files contain the stored values in a binary format.

File name rules: data_<resolution>_<timestamp>.bin. <resolution> is a number (0 to 7) that specifies the resolution of the data contained in this file; seeResolution. <timestamp> is the start time of this file. The timestamp format is "yyyymmddhhmm" (4 digits for year, 2 digits each for month, day, hour and minute).

All values are stored in the little-endian format (least significant byte first). Each file contains a number of records. The length of each record is determined by the settings in Var.ini.

Each entry has a 16-byte header with the following fields:

tv_sec
unsigned int64; Timestamp, given as seconds since 1970-01-01 00:00 UTC
tv_nsec
unsigned int32; Timestamp, nanoseconds in the second given by tv_sec
quality
unsigned int32; Quality for the stored value

After the header follow the user data. After the user data (without any padding) follows the next header.

Live data format (resolution 0)

These are the unmodified values as given by OPC or written by an application. Data length is given by the DataType entry in Var.ini, multiplied by the ArrayLength entry (1 if not present).

DataType=string: Each string has a 4-byte header (giving the really used number of bytes) followed by the data bytes (always ElementLength bytes).

DataType=struct: Each structure has a length as given in ElementLength

Structure encoding rules

- "bit" elements occupy 1 bit.
- all numeric elements occupy as many bytes as needed (documented below).
- "string" elements have a 4-byte length element followed by the character data (usually UTF-8 encoded). If a maximum string length is given, the character data are padded to this length with "\x00" bytes to ensure a fixed structure layout. Othweise no implicit padding exists.
- "time" elements occupy 12 bytes: 8 byte UNIX-Timestamp (seconds since 1970-01-01 00:00:00 UTC) and 4 byte Nanoseconds
- "struct" elements occupy as many bytes as required for their encoding
- all elements except "bit" start at a byte boundary (align 1). Padding bytes (if required) must be specified explicitly via PADDING elements
- "bit" elements start at a byte boundary if:
  - they have an Array Size other than 1 or VARIABLE_ARRAY_LENGTH is set
  - and PACKED is not set
- "bit" elements are packed in one byte if:
  - the Array Size is 1 and VARIABLE_ARRAY_LENGTH is not set
  - or PACKED is set
- If required, implicit padding bits are inserted after "bit" elements.

Aggregate data format (resolutions 1 - 7)

These are aggregated values calculated during data recording. Data length is determined by the selected aggregate functions. The following fields may be present (in the given sequence):

Field	Min	Max	Count	Avg	StdDev	Type	Description
Minimum	X	-	-	-	-	DataType AggregateCalcDataType if DataType=struct	The minimal value in the interval.
Maximum	-	X	-	-	-	DataType AggregateCalcDataType if DataType=struct	The maximal value in the interval.
Count	-	-	X	-	-	always unsigned int64	The number of data changes in the interval.
Sum	-	-	-	X	X	always double	The weighted sum of all data in the interval.

Field	Min	Max	Count	Avg	StdDev	Type	Description
tv_sec	-	-	-	X	X	always unsigned int64	The number of seconds of data recorded in the interval.
tv_nsec	-	-	-	X	X	always unsigned int32	The number of nanoseconds after the last second of data recorded in the interval.
SumSq	-	-	-	-	X	always double	The sum of all squared weighted data values in the interval.

To calculate the average value in the interval: Avg = Sum / (tv_sec + tv_nsec / 1000000000)

To calculate the standard derivation in the interval: StdDev = sqrt(SumSq / (tv_sec + tv_nsec / 1000000000) - Avg * Avg)

Data types

Name	Size	Description
bit	1 Bit	truth value (false or true)
u8, u16, u32, u64	1-8 bytes	unsigned integral types with 8-64 bit (1-8 bytes)
i8, i16, i32, i64	1-8 bytes	signed integral types with 8-64 bit (1-8 bytes)
f32, f64	4 or 8 bytes	IEEE-floating-point values with 32 or 64 bit (4 or 8 bytes)
string	variable	character string, UTF-8 encoded
time	12 byte	UNIX timestamp, with resolution 1 nanosecond
struct	variable	structured data type, encoded according to this rules
enum	4 byte	enumeration, always encoded as u32

Resolution values

The following resolutions are available for data acquisation:

Fastest
: Save values as fast as the source generates them

Second
: Save at most one value per second

Minute
: Save at most one value per minute

Hour
: Save at most one value per hour

Day
: Save at most one value per day

The following resolutions are available for file rotation:

Hour
: Start a new file at the beginning of each hour (UTC time): 2000-01-01 00:00, 2000-01-01 01:00, 2000-01-01 02:00, ...

Day
: Start a new file at midnight (UTC time) of each day: 2000-01-01 00:00, 2000-01-02 00:00, 2000-01-03 00:00

Week
: Start a new file at midnight (UTC time) of each monday: 2000-01-03 00:00, 2000-01-10 00:00, 2000-01-17 00:00, 00

Month
: Start a new file at midnight (UTC time) of the first day of each month: 2000-01-01 00:00, 2000-02-01 00:00, 2000-03-01 00:00, ...

Year
: Start a new file at midnight (UTC time) of january 1st: 2000-01-01 00:00, 2001-01-01 00:00, 2002-01-01 00:00, ...

Historian REST interface

The HistorianAPI is a Websocket Interface that transfers JSON data. By default it is listening on port 8083. This can be changed by editing ConfigServer.Settings.

All API requests expect an UTF-8 encoded JSON object. The following properties are used for all requests:

- "function": string, required. Possible are:
  - Historian/Read
  - Historian/Write
  - Historian/AddVariable
  - Historian/DeleteVariable
  - Historian/ReadVariable
  - Historian/ListVariables
  - Historian/ReadGlobalSettings
  - Historian/WriteGlobalSettings
  - Historian/ReadChangeTimestamp
  - Historian/DiagVariable
  - Historian/ReadStruct
  - Historian/WriteStruct
  - Historian/ListStructs
  - Historian/DeleteStruct
- "id": string or number, optional, returned unmodified to allow identifying the answer

More properties may exist, depending on the function to execute. Any unknown properties are silently ignored. For future compatibility avoid sending undocumented properties.

All API answers are returned as an UTF-8 encoded JSON object. The following properties always exist:

- "function": string, copied from request
- "id": string or number, copied from request if present
- "status": number, status code of request (0 is OK, more values aredocumented below)

More properties may exist, depending on the function to execute. For future compatibility, the client should ignore any properties it does not recognize.

Function "Historian/Read"

Reads historical values.

Additional request properties:

- "variable": string, required, name of the historian variable to read
- "subvariable": string, optional, name of the sub variable for group variables
- "valuecount": number, optional, maximum number of values to return
- "start", "stop": string, optional, ISO-8601-formatted date, bounds of requested timespan (documented below)
- "resolution": string, required, requested time resolution (documented below)
- "aggregate": string, required, requested aggregate function (documented below)
- "includebounds": boolean, optional, true to include bounds values (one value before and after the specified interval, unless the given bound exactly matches a value); default false
- "reverse": boolean, optional, true to return values in reverse order (newest value first); default false

Notes:

- At least 2 of "start", "stop" and "valuecount" must be given.
- If both "start" and "stop" are given, "start" must not be earlier than "stop".

Additional response properties:

- "variable": string, copied from request
- "subvariable": string, copied from request (if present)
- "values": array of objects, values returned (documented below)
- "blocked": boolean, true if more values exist than could be returned in this answer (limited by "valuecount" or memory constraints)

example request:

```
{
 "function": "Historian/Read",
 "id": 12345,
 "variable": "New Variable",
 "valuecount": 100,
 "start": "2021-04-20T11:30:00Z",
 "stop": "2021-04-20T11:30:10Z",
 "resolution": "second",
 "aggregate": "avg"
}
```

example response:

```
{
 "function": "Historian/Read",
  "id": 12345,
  "status": 0,
  "variable": "New Variable",
  "values": [
   { "quality": 0, "time": "2021-04-20T11:30:00Z", "value": 10 },
   { "quality": 0, "time": "2021-04-20T11:30:02Z", "value": 11 },
   { "quality": 0, "time": "2021-04-20T11:30:04Z", "value": 12 },
   { "quality": 0, "time": "2021-04-20T11:30:06Z", "value": 13 },
   { "quality": 0, "time": "2021-04-20T11:30:08Z", "value": 14 },
   { "quality": 0, "time": "2021-04-20T11:30:10Z", "value": 15 },
  ],
 "blocked": false
}
```

Function "Historian/Write"

Writes historical values. The Historian only supports adding values in sequential order. Values that are passed out of sequence are silently ignored.

Additional request properties:

- "variable": string, required, name of the historian variable to write
- "values": array of objects, required, values to write (documented below)

Notes:

- For variable groups, each element of "values" has an object as "value", which maps subvariable names to values

Additional response properties:

- "variable": string, copied from request

example request:

```
{
 "function": "Historian/Write",
 "id": 12345,
 "variable": "New Variable",
 "values": [
  { "quality": 0, "time": "2021-04-20T11:30:00Z", "value": 10 },
  { "quality": 0, "time": "2021-04-20T11:30:02Z", "value": 11 },
  { "quality": 0, "time": "2021-04-20T11:30:04Z", "value": 12 },
  { "quality": 0, "time": "2021-04-20T11:30:06Z", "value": 13 },
  { "quality": 0, "time": "2021-04-20T11:30:08Z", "value": 14 },
  { "quality": 0, "time": "2021-04-20T11:30:10Z", "value": 15 },
 ]
}
```

example response:/

```
{
 "function": "Historian/Write",
 "id": 12345,
 "status": 0,
```

```
 "variable": "New Variable"
}
```

Function "Historian/AddVariable"

Adds a new variable to the Historian. Variables are identified by a unique name. Calling this function with a name that already exists allows changing some properties. When trying to change properties that are not changeable (especially anything that affects file storage format), an error occurs.

Additional request properties:

- "variable": object, required, describes the variable to add/modify (documented below)

Additional response properties:

- "variable": string, the variable name, copied from request

example request:

```
{
 "function": "Historian/AddVariable",
 "id": 12345,
 "variable":
 {
  "name": "New Variable",
  "type": "UInt16",
  "enabled": true,
  "opc_enabled": true,
  "opc_conn": "S7-1500",
  "opc_variable": "Datablocks.OPC_DB.Temperature",
  "opc_resolution": "second",
  "file_save": false,
  "mem_size": 100
 }
}
```

example response:

```
{
 "function": "Historian/AddVariable",
 "id": 12345,
 "status": 0,
 "variable": "New Variable"
}
```

Function "Historian/DeleteVariable"

Deletes a variable from the Historian. In case of file logging, the data files are not removed.

Additional request properties:

- "variable": string, required, the variable name

Additional response properties:

- "variable": string, the variable name, copied from request

example request:

```
{
 "function": "Historian/DeleteVariable",
 "id": 12345,
 "variable": "New Variable"
}
```

example response:

```
{
 "function": "Historian/DeleteVariable",
 "id": 12345,
 "status": 0,
 "variable": "New Variable"
}
```

Function "Historian/ReadVariable"

Reads a variable configuration from the Historian.

Additional request properties:

- "variable": string, required, the variable name

Additional response properties:

- "variable": object, the variable configuration (documented below)

example request:

```
{
 "function": "Historian/ReadVariable",
 "id": 12345,
 "variable": "New Variable"
}
```

example response:

```
{
```

```
 "function": "Historian/ReadVariable",
 "id": 12345,
 "status": 0,
 "variable":
 {
  "name": "New Variable",
  "type": "UInt16",
  "enabled": true,
  "opc_enabled": true,
  "opc_conn": "S7-1500",
  "opc_variable": "Datablocks.OPC_DB.Temperature",
  "opc_resolution": "second",
  "file_save": false,
  "mem_size": 100
 }
}
```


Function "Historian/ListVariables"

Reads the list of variable configurations from the Historian.

Additional request properties:

- "startoffset": number, optional, number of first returned variable; 0 if not given
- "variablecount": number, optional, maximum number of variables to return, 100 if not given

Additional response properties:

- "variables": array of object, the variable configurations (documented below)
- "blocked": boolean, true if more variables exist than could be returned in this answer (limited by "variablecount" or memory constraints)

example request:


```
{
 "function": "Historian/ListVariables",
 "id": 12345,
 "startoffset": 0,
 "variablecount": 100
}
```


example response:


```
{
 "function": "Historian/ListVariables",
 "id": 12345,
 "status": 0,
 "variables": [
 {
  "name": "New Variable",
  "type": "UInt16",
  "enabled": true,
  "opc_enabled": true,
  "opc_conn": "S7-1500",
  "opc_variable": "Datablocks.OPC_DB.Temperature",
  "opc_resolution": "second",
  "file_save": false,
  "mem_size": 100
 }
 ],
 "blocked": false
}
```


Function "Historian/ReadGlobalSettings"

Reads the global settings object of the historian

Additional request properties:

- none

Additional response properties:

- "mindiskspace": number, minimum free disk space (in MiB)

example request:


```
{
 "function": "Historian/ReadGlobalSettings",
 "id": 12345
}
```


example response:


```
{
 "function": "Historian/ReadGlobalSettings",
 "id": 12345,
 "status": 0,
 "mindiskspace": 1000
}
```


Function "Historian/WriteGlobalSettings"

Writes the global settings object of the historian

Additional request properties:

- "mindiskspace": number, minimum free disk space (in MiB)

Additional response properties:

- none

example request:

```
{
 "function": "Historian/WriteGlobalSettings",
 "id": 12345,
 "mindiskspace": 1000
}
```

example response:

```
{
 "function": "Historian/WriteGlobalSettings",
 "id": 12345,
 "status": 0
}
```

Function "Historian/ReadChangeTimestamp"

Reads the timestamp of the last historian variable change (addition, modification or deletion).

Additional request properties:

- none

Additional response properties:

- "time": string, ISO-8601-formatted timestamp (documented below)

example request: {
 "function": "Historian/ReadChangeTimestamp",
 "id": 12345
}
example response: {
 "function": "Historian/ReadChangeTimestamp",
 "id": 12345,
 "status": 0,
 "time": "2021-04-20T11:30:00Z"
}

Function "Historian/DiagVariable"

Reads diagnostic information for a historian variable.

Additional request properties:

- "variable": string, required, the variable name

Additional response properties:

- "variable": string, the variable name, copied from request
- "starttime": string, ISO-8601-formatted timestamp (documented below), timestamp of oldest available value
- "currenttime": string, ISO-8601-formatted timestamp (documented below), timestamp of most recent value
- "currentquality": number, quality code of the most recent value (0 is OK, more values aredocumented below)
- "writeerror": number, error code of the most recent disk write operation (0 is OK, more values aredocumented below)

example request:

```
{
 "function": "Historian/DiagVariable",
 "id": 12345,
 "variable": "New Variable"
}
```

example response:

```
{
 "function": "Historian/DiagVariable",
 "id": 12345,
 "status": 0,
 "variable": "New Variable",
 "starttime": "2021-04-20T11:30:00Z",
 "currenttime": "2021-04-20T11:30:10Z",
 "currentquality": 0
}
```

Function "Historian/ReadStruct"

Reads structure information for a historian variable. Can be used to decode structured data.

Additional request properties:

- "variable": string, required, the variable name
- "structname": string, required, the structure name

Additional response properties:

- "variable": string, the variable name, copied from request
- "structname": string, the structure name, copied from request
- "structcomment": string, optional, the structure comment (if present)
- "structdata": array of objects, the structure elements (documented below)

example request:


```
{
 "function": "Historian/ReadStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure"
}
```


example response:


```
{
 "function": "Historian/ReadStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure",
 "structcomment": "commment text ...",
 "structdata": [
  {
   "name": "element 1",
   "type": "uint8",
   "arraylength": 10
  },
  {
   "name": "element 2",
   "type": "uint32"
  }
 ]
}
```


Function "Historian/WriteStruct"

Writes structure information for a historian variable. The Historian doesn't use this, but it can be read back via "Historian/ReadStruct".

Additional request properties:

- "variable": string, required, the variable name
- "structname": string, required, the structure name
- "structcomment": string, optional, the structure comment
- "structdata": array of objects, the structure elements (documented below)

Additional response properties:

- "variable": string, the variable name, copied from request
- "structname": string, the structure name, copied from request

example request:


```
{
 "function": "Historian/WriteStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure",
 "structcomment": "commment text ...",
 "structdata": [
  {
   "name": "element 1",
   "type": "uint8",
   "arraylength": 10
  },
  {
   "name": "element 2",
   "type": "uint32"
  }
 ]
}
```


example response:


```
{
 "function": "Historian/WriteStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure"
}
```


Historian/ListStructs

Lists structure information for a historian variable.

Additional request properties:

- "variable": string, required, the variable name
- "startoffset": number, optional, the start offset; 0 if not given

Additional response properties:

- "variable": string, the variable name, copied from request
- "structs": array of objects:
  - "structname": string, the structure name
  - "structcomment": string, optional, the structure comment (if present)
  - "structdata": array of objects, the structure elements (documented below)
- "blocked": boolean, true if more structures exist than could be returned in this answer (limited by memory constraints)

example request:

```
{
 "function": "Historian/ListStructs",
 "id": 12345,
 "variable": "New Variable",
 "startoffset": 0
}
```

example response:

```
{
 "function": "Historian/ListStructs",
 "id": 12345,
 "variable": "New Variable",
 "structs": [
  {
   "structname": "New Structure",
   "structcomment": "commment text ...",
   "structdata": [
    {
     "name": "element 1",
     "type": "uint8",
     "arraylength": 10
    },
    {
     "name": "element 2",
     "type": "uint32"
    }
   ]
  }
 ]
}
```

Function "Historian/DeleteStruct"

Deletes structure information for a historian variable.

Additional request properties:

- "variable": string, required, the variable name
- "structname": string, optional, the structure name; if not given, all structures are deleted

Additional response properties:

- "variable": string, the variable name, copied from request
- "structname": string, the structure name, copied from request (if present)

example request:

```
{
 "function": "Historian/DeleteStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure"
}
```

example response:

```
{
 "function": "Historian/DeleteStruct",
 "id": 12345,
 "variable": "New Variable",
 "structname": "New Structure"
}
```

value objects

properties:

- "quality": number, optional, quality code of the value, 0 if not given (0 is OK, more values aredocumented below)
- "time": string, optional, ISO-8601-formatted timestamp (documented below) current time if not given
- "value": any type, data value; format depends on data type of variable
  - boolean: boolean
  - int8/16/32/64, uint8/16/32/64: number, fractional values not allowed
  - float, double: number, optinally including fractions
  - string: string
  - timestamp: string (ISO-8601 time format)
  - structure: array of numbers (0-255), binary structure data
  - or an array of the values above

example:

```
{ "quality": 0, "time": "2021-04-20T11:30:00Z", "value": 10 }
```

variable object

general properties:

- "name": string, required, unique name of the variable
- "type": string, optional, data type of the variable data (documented below)
- "structname": string, required if "type" is "structure", name of structure; ignored otherwise
- "arraylength": number, optional, length of array; if not given, 1 is assumed
- "elementlength": number, required if "type" is "string" or "structure" length of one array element, given in bytes; ignored otherwise
- "enabled": boolean, optional, true if variable is enabled (uses memory and license), false if variable is disabled (doesn't work, does not use memory or license, can be reenabled if needed); default false

- "sub_variables": array of object, required unless "type" is given, list of sub variables to combine in one historian variable
  - "name": string, required, unique name of the variable
  - "type": string, required, data type of the variable data (documented below)
    Note: "structure" is not supoprted
  - "elementlength": number, required if "type" is "string" or "structure" length of one array element, given in bytes; ignored otherwise
  - "opc_conn": string, required if "opc_enabled" is true, name of connection to read from
  - "opc_group": string, optional, group name of connection to read from
  - "opc_variable": string, required if "opc_enabled" is true, name of variable to read from

properties for OPC data acquisation:

- "opc_enabled": boolean, optional, true if OPC data acquisation is enabled; default false
- "opc_conn": string, required if "opc_enabled" is true and "type" is given, name of connection to read from
- "opc_group": string, optional, group name of connection to read from
- "opc_variable": string, required if "opc_enabled" is true and "type" is given, name of variable to read from
- "opc_resolution": string, required if "opc_enabled" is true, resolution for OPC reading (documented below)

properties for data storage:

- "file_save": boolean, optional, true to save to file; default false
- "aggregates": array of string, optional, list of aggregates to save (documented below)
- "aggregate_calcoffset": number, optional, byte-offset (if "type" is "structure") or index (if "type" is a simple type and "arraylength" is > 1) of the data element that is used for aggregate calculations; default 0
- "aggregate_type": string, optional, data type used for aggregate calculation (if "type" is "structure"); default uint8; ignored otherwise
- "file_resolution": string, required if "file_save" is true, time after which a new data file is started (documented below)
- "file_maxcount": number, optional, maximum number of data files to keep; default 0 = unlimited
- "file_mindiskspace": number, optional, minimum free disk space (in MiB); default 0 = use global setting
- "mem_size": number, required if "file_save" is false, size of in-memory storage (in MiB)

example:

```
{
 "name": "New Variable",
 "type": "uint16",
 "enabled": true,
 "opc_enabled": true,
 "opc_conn": "S7-1500",
 "opc_variable": "Datablocks.OPC_DB.Temperature",
 "opc_resolution": "second",
 "file_save": false,
 "mem_size": 100
}
```

structure element object

properties:

- "name": string, required, the element name
- "comment": string, optional, the element comment
- "type": string, required, data type of the structure element (documented below)
- "structname": string, required if "type" is "structure", name of structure; ignored otherwise
- "arraylength": number, optional, length of array; if not given, 1 is assumed
- "elementlength": number, optional; if "type" is "string": maximum length of one string, given in bytes, no fixed length if not given or 0; ignored otherwise
- "lengthfieldname": string, optional, the element name of the element that gives a dynamic array length; this oferrides "arraylength"; the length field must have a numeric type
- "switchfieldname": string, optional, the element name of the element that marks this element as "present" or "absent"; the switch fiels must have a numeric or boolean type
- "switchvalue": number or boolean, optional, the value of the "switchfield" that marks this element as "present"
- "packed": boolean, optional, true if multiple boolean fields should be packed together in one byte; default false
- "padding": boolean, true if this field is a padding field that contains no useful data; default false

structure layout rules:

- "boolean" elements occupy 1 bit.
- all numeric elements occupy as many bytes as needed (documented below).
- "string" elements have a 4-byte length element followed by the character data (usually UTF-8 encoded). If "elementlength" is given, the character data are padded to this length with "\x00" bytes to ensure a fixed structure layout. Othweise no implicit padding exists.
- "time" elements occupy 12 bytes: 8 byte UNIX-Timestamp (seconds since 1970-01-01 00:00:00 UTC) and 4 byte Nanoseconds
- "structure" elements occupy as many bytes as required for their encoding
- all elements except "boolean" start at a byte boundary (align 1). Padding bytes (if required) must be specified explicitly via "padding": true
- "boolean" elements start at a byte boundary if:
  - they have an "arraylength" other than 1 or a "lengthfieldname"
  - and "packed" is false (or unspecified)
- "boolean" elements are packed in one byte if:
  - the "arraylength" is unspecified or 1 and no "lengthfieldname" exists
  - or "packed" is true
- If required, implicit padding bits are inserted after "boolean" elements.

resolution values

- "maximum": save/return all available values
- "second": save/return one value per second
- "minute": save/return one value per minute
- "hour": save/return one value per hour
- "day": save/return one value per day
- "week": save/return one value per week
- "month": save/return one value per month
- "year": save/return one value per year

aggregate values

- "value": return actual values (for lower resolutions: firat value of the interval)

- "min": return minimum value in given interval
- "max": return maximum value in given interval
- "avg": return average value in given interval
- "count": return number of saved value in given interval

ISO-8601 time format

note: this is only a subset of ISO-8601

```
2021-04-20T11:30:10.123+00:00
^    ^  ^  ^  ^  ^  ^ ^------- optional Timezone offset, allowed are "Z" (UTC)
|    |  |  |  |  |  |          or a zone difference in hours (and optionally
|    |  |  |  |  |  |          minutes), positive is east, UTC if not given
|    |  |  |  |  |  +---------- optional second fractions, up to 9 digits are
|    |  |  |  |  |             supported
|    |  |  |  |  +------------- second, always 2 digits
|    |  |  |  +---------------- minute, always 2 digits
|    |  |  +------------------- hour, always 2 digits, 24-hour format
|    |  +---------------------- day, always 2 digits
|    +------------------------- month, always 2 digits
+------------------------------ year, always 4 digits
```

Other ISO-8601 formats (e.g. week numbers, day-of-year, ...) are not supported. All returned timestamp values are in UTC.

Data types

- "boolean": truth value (false or true)
- "uint8", "uint16", "uint32", "uint64": unsigned integral types with 8-64 bit (1-8 bytes)
- "int8", "int16", "int32", "int64": signed integral types with 8-64 bit (1-8 bytes)
- "float", "double": IEEE-floating-point values with 32 or 64 bit (4 or 8 bytes)
- "string": character string, UTF-8 encoded
- "time": timestamp, transferred on JSON API as ISO-8601 string
- "structure": structured data type, transferred on JSON API as array of uint8

Status codes

- 0: OK
- 102: No Variable - the given variable name was not found
- 103: Out of Memory - a memory allocation failed
- 108: Not Ready - a valid but unimplemented function was called
- 143: Bad Opcode - an unknown function was called
- 144: Bad Length - a value given has a wrong length
- 1304: Disk Full - no space left on disk, or the allocated quota is exceeded
- 1307: Access Denied - the requested operation is not allowed
- 1308: No write access - the write operation failed
- 1319: Invalid Value - a value given is wrong
- 1324: License Limit - the operation would exceed a license limit

Other numbers signal an unspecified or undocumented error. Please contact TANI support in such cases.

Quality codes

- 0: Good - variable value is valid
- 1: Configuration Error - a configuration problem exists (e.g a missing connection)
- 2: Not Connected - the connection could not be established
- 3: Device Failure - the communication target device failed
- 4: Sensor Failure - the value to read is invalid or missing
- 5: Last Known Value - a temporary communication problem prevents fetching new values
- 6: Communication Failure - an unknown communication error occurted (e.g. wrong protocol)
- 7: Out Of Service - the communication target is temporarily not available
- 8: Waiting for Initial Data - the state after connection establish until the first valid data arrives
- 9: Write Protected - (for write operation only) the value can not be written
- 10: Corrected Value - the value was abnormal or out of range and has been corrected
- 12: Read Protected - (for read operations only) the value can not be read
- 13: Convert Failed - a type conversion failed
- 14: Overload - not enough resources to process the request (e.g out of memory)

Other numbers signal an unspecified or undocumented error. Please contact TANI support in such cases.