TCPBinaryDriver

A binary mode tcp OPC-UA driver.

- This module is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License
- This module may cause the Ignition(tm) Software to fail to function correctly or function at all. An update of Ignition(tm) may cause this Module to fail to function correctly or function at all. The author shall have no liability whatsoever for any failure after the installation of this module or after the installation of a Software update.
- This module is licensed under the Apache License, Version 2.0.

Overview

This driver connects devices that send messages with binary data to the OPC-UA server. The message format is configurable for each device. This driver does not poll data but expects the connected device to send messages on its own.

Possible applications are:

- Connect devices that are not supported by default drivers but allow custom TCP-IP connection.
- Log data from devices when an exact source timestamp is needed.
- Log data from devices that use internal data buffering.

Driver Settings

General

- **Device Name**
  The device name. If this name is changed, any queued messages in queue mode will get lost!

Connectivity

- **Hostname**
  Hostname or IP address of the device.
- **Port**
  The port to connect to.
- **Connect on startup**
  If false, the driver will not connect on startup, but waits for a client to set '[Status]/Connect' to true. Use this setting if you want to dynamically set the IP address or port from a client (see Special Tags).
- **Message Timeout**
  Amount of time (in milliseconds) without receiving data from the device before a disconnect/reconnect is made. Set to 0 to disable automatic reconnection. As this driver does not poll from the device, it will not notice a disrupted connection. It is recommended to have the

device send a periodical watchdog message. A special message with only a dummy tag is sufficient for this purpose.

Message handling

- **Reverse byte order**
  Set true to use LittleEndian (Intel) byte order instead of default BigEndian (Motorola).
- **Timestamp factor**
  A multiplier for the timestamp / message age received from the device. The driver expects those values to be in milliseconds.
  If the device, for example, provides the message age in seconds, use a factor of 1,000 to convert this to milliseconds.

Message configuration

General Settings

- **Type of message ID** - The message id sent by the device.
  - None - Only a single message configured with ID 0 is used.
  - Byte - Message ID is sent as a byte. Message ID's 0..255 can be used.
  - Word - Message ID is sent as a word. Message ID's 0..65535 can be used.

Message specific Settings

- **Queue Mode** - The queue mode is useful in situations, where the device sends messages faster than they can be evaluated by the client (see Queue Mode).
  - None - Message queue is not used.
  - Handshake - Messages are stored in a queue and published to the client one by one when the Handshake tag is set.
  - Delayed - Mesages are stored in a queue and published with a fixed delay for each essage. The delay is 3 times the fastest subscription rate to this messages. If there are no subscriptions to `_Timestamp` or `_MessageCount`, messages are not published.
- **Use Persistance** - If true, the queue file is stored to the gateways disk on every change, to prevent data loss. If false, the queue is only saved on a regular shutdown or driver restart.

Tag configuration

The tags contained in a message can be freely configured. Up to 254 tags are supported per Message. The first row of the config table is used to set message id and message alias. Subsequent rows are used to configure the tags.

- **Offset** - The byte offset in the message - Only for information.
- **Data Type** - The tags data type (see below)
- **ID** - A unique tag id (1..254)
- **Alias** - The tags alias - used as address and browse name.
- **Size** - For String tags, this is the length of the string. With all other data types, an array of the given size is created. The driver creates a single child tag for every array element, the parent tag contains the complete array. For OPC-UA clients that don't support arrays (like Ignition™) the parent elements are useless (although they may be coerced to a string).

Data Types

- **Dummy**
  A byte at this position is simply ignored.
- **Unsigned Byte**
  Unsigned Byte (0..255)
- **Signed Byte**
  Signed Byte (-128..127)
- **Bool8**
  A byte treated as 8 single bits. The single bits are addressed by adding [nn] to the tags address where nn is the bit number. The special element [raw] contains the bytes value.
- **Bool16**
  Same as Bool8 but with 16 bits.
- **UInt16**
  An unsigned 16 bit integer.
- **Int16**
  An signed 16 Bit integer
- **UInt32**
  An signed 32 bit integer.
- **Int32**
  A signed 32 bit integer.
- **Float**
  A 32 bit float according to IEEE 754 floating-point "single format".
- **String**
  The received data is decoded as a ISO-8859-1 string, leading and trailing whitespaces and special chars are removed using the Java String#trim method.
  The `size` parameter defines the length of the string, arrays of strings are not supported.
- **Message Age**
  An unsigned 32 bit Integer with the absolute age of the message or a relative timestamp if a header with a packet timestamp is configured. This value is used to calculate the special `_Timestamp` tag and the timestamps of all tags in this message.
  The incoming value is multiplied by the Timestamp factor to get a millisecond resolution. The published OPC tag `_Message Age` contains the calculated value.

Addressing

The address of a driver tag is `messageAlias/tagAlias[Index][Bit]`.
Example:
`Message1/Tag1[0]` - Bit 0 in Tag1 in Message1
`Message1/Tag2` - Tag2 in Message1

Note: In the browse tree, the square brackets are replaced by a underscore to support drag'n'drop with clients that don't allow brackets in the tag name.

Special Tags

Status Folder

The folder [Status] contains some tags to control the device connection:

- `Device Name` - The device name as configured in the gateway.

- `Is connected` - True, if the driver is connected to the device.
- `Connected Host` - IP address and hostname of the connected device.
- `Connect` - Writable tag to control the connection state. Writing 'false' will disconnect from the device.
- `Hostname` - Writable tag to set hostname or IP-address. Only effective if Connect is false.
- `Port` - Writable tag to set the port to connect to. Only effective if Connect is false.

Example script for changing hostname or port:
```
system.opc.writeValue('Ignition OPC-UA Server', '[TCPBin][Status]/Connect', 0)
system.opc.writeValue('Ignition OPC-UA Server', '[TCPBin][Status]/Hostname', 'device.example.org')
system.opc.writeValue('Ignition OPC-UA Server', '[TCPBin][Status]/Port', 2001) system.opc.writeValue('Ignition OPC-UA Server', '[TCPBin][Status]/Connect', 1)
```

Special tags in message folders

The driver generates some special tags for each configured message:

- `_Timestamp` - Time when the current message was received from the device, maybe corrected by a configured `MessageAge` tag.
  `Timestamp = ReceiveTime - MessageAgeTag [+ PacketTimestamp]`
- `_MessageCount` - Total count of messages published to clients. In Handshake Mode this is the count of published messages, without messages still waiting in the buffer.
  In redundant systems, this tag will be reset to 0 when the active node changes. If using the queue mode 'Handshake' with a transaction group, it is recommended to use this tag as a trigger, because the current message may be republished after a failover, so the timestamp or values may not change in this situation.
- `_Handshake` - Writable handshake for synchronisation (see Queue Mode). Only allowed value to write is 'True'.
- `_QueueSize` - Count of queued messages in handshake or delayed mode

Header

Configuration

- **Use Header** - If true, all incoming data packets have to start with this header. If false, a header is not used.
- **Packet size includes header** - If true, the packet size received with the header includes the length of the header itself.
- **Use Handshake Message** - If true, a handshake message is send back to the device after all messages in the current have been evaluated and added to the queue in queue mode.
- **Handshake message** - The handshake message to send. A comma seperated list of bytes. Decimal, hexadecimal, and octal numbers are accepted. Allowed decimal range is -128...255. Accepted seperators are comma, semicolon and space.
  The strings 'timestamp' and 'sequence' are replaced with the received value, 'lenb' (Byte) or 'lenw' are replaced with the message length.
  Example: '`lenw, 0xff,0xfe,timestamp,sequence`'

Header Data Types

- **Packet Size**
  A 16bit value containing the size of the packet in bytes.
- **Packet Timestamp** A 32bit timestamp. Used in combination with a MessageAge tag to calculate the age of individual messages in the packet.
  This mechanism is useful if the connected device does not send data immediately but has some kind of internal storage. Every time a message is stored, the devices stores an internal timestamp in this message. When the stored messages are sent later on, the device updates the packet timestamp in the header with the current value of the same internal timestamp.
  When receiving the packet, the driver calculates the age of each individual message as the difference between the two timestamps (PacketTimestamp-MessageAge).
  Note that the device timestamps does not have to be a real 'time'. You can also use internal uptime counters or time ticks, which should be available even if the connected device does not have a real time clock.
- **Sequence Id**
  A sequence id send by the device. This type is intended to be used in the handshake message.
- **Fixed Byte**A byte with the given value has to be present in the header to indicate a valid packet. If the received header contains any other value at this position, the complete packet is discarded. It is recommended to supply a packet size if this may happen.
  The value is interpreted as hexadecimal if it starts with '0x'.
- **Fixed Word**
  Same as fixed byte, but checks a 16 bit value.
- **Dummy**
  The number of bytes given in 'Size' is simply ignored.

Writeback

The driver supports a simple writeback mechanism to write data back to the plc. The main purpose of this function is to request data updates from the plc, it is not designed for transferring large amounts of data (though this is possibly using the ByteString data type).

Configuration

- **Use message ID** - Select `Byte` or `Word` to send a message id. The message id is unsigned, negative values are not accepted. The message id is only send if it is configured in the Prefix.
- **Use Prefix** - If true, the configured prefix will be send at the beginning of the outgoing message
- **Prefix** - A comma seperated list of bytes. Decimal, hexadecimal, and octal numbers are accepted. To send the message length use the strings '`lenb`' (Byte) or '`lenw`' (Word).
  The String '`id`' is replaced by the message Id set by the client.
  Example: '`lenb, 0xfe, 0xfe, id`'
- **Data Type** - The data type to send.
  - UInt16 - Unsigned 16bit Integer
  - Int16 - Signed 16bit Integer
  - UInt32 - Unsigned 32bit Integer
  - Int32 - Signed 32bit Integer
  - ByteString - A comma seperated list of bytes. Decimal, hexadecimal, and octal numbers are accepted. Allowed decimal range is -128...255. Accepted seperators are comma, semicolon and space.
    Example: `0xff,255,-127,127` will be parsed as '0xff,0xff,0x81,0x7f'
- **Send on value change** - If true, the driver will send a message when the client changes the value. If false, send must be triggered explicitly with the special 'Write' tag.

- **Send initial value** - Sends an initial value as soon as the connection to the device has been established. This function may be used to request initial values from the connected device.
- **Initial value** - The value to send after connection to the device has been established.
- **Initial ID** - The ID to send after connection to the device has been established.
  The initial id and the initial value are set 'silently', an id or value that has been set by a client will not change.

Queue Mode

This driver does not poll, but receives messages send by the connected device. If the device sends messages at a faster rate than the publishing interval used by the OPC-Client (in Ignition™ this is the rate of the scan class), you will loose data. It is up to the programmer of the connected device to send data at a reasonable rate.
For situations where a device sends a burst of messages, Handshake or Delayed queue modes can be used. In this mode, the driver buffers incoming messages in memory and publishes them one by one to the client.
To prevent message loss in case of an server failure, buffered messages are stored to the gateway's disk (data/drivers/TcpBin_...) if the message is configured to use persistance. They are reloaded on startup, unless the message configuration has been changed. The storage mechanism used is a compromise between performance and data safety. The file handles are kept open all the time, but the file is flushed after each write. The file is also periodically rewritten completely, so there are (minimal) time spans where a failure will result in complete data loss.
The message is removed from the queue only after the handshake is set by a client or the delay is expired. If a shutdown happens during this period, the same message will be published again after the restart. This behaviour ensures that no data is lost, but requires additional actions to prevent double logging (e.g. using the timestamp as a unique key.) It also means that values or timestamp should not be used as triggers.

Important!
Even if using the persistent queue mode or redundant configurations, there are situations where data **will be lost!**
Do not use this driver if this is not acceptable!

Handshake Mode

In this mode a special witeable tag `_Handshake` is added to the message. Once the handshake is set by the client, the driver publishes the next message and resets the handshake tag.
The client should not use the `_Handshake` tag to detect a new message but use the `Timestamp` or `MessageCount` tag instead.
Please note that the driver will not publish new messages if the client does not set the handshake. It may be a bad idea to use the handshake mode with Ignitions™ default transaction group handshake.

Delayed Mode

In this mode incoming messages are published to clients with a fixed delay. This delay is 3 times the fastest subscription rate for the respective message, so make sure to subscribe at least with about the triple rate of expected data changes.
Publishing of messages in delayed mode starts when one of tags `_Timestamp` or `_MessageCount` are subscribed by a client. If there are no subscriptions, no messages are published. This makes sure that a client is connected before the driver start publishing messages.

Redundancy

In redundant systems, only the active master gateway will connect to the device. Writable tags, e.g. the value/message id of the writeback folder and the current driver state (connected, ip and port), are mirrored to the backup gateway.
After a switchover to the backup gateway, the backup will connect to the device, if the master gateway was connected. Initial values will be send to the device if this is configured in the writeback section.
In Delayed or Handshake Queue mode, the active message may be republished after a failover. Keep this in mind when configuring 'On Change' trigger settings.

Important!
Even if using the persistent queue mode or redundant configurations, there are situations where data **will be lost!**
Do not use this driver if this is not acceptable!

Remarks

The DriverStatus on the Gateway Status Panels always shows 'Connected', even if the device is offline. This is necessary to make Subscriptions in Handshake mode and the Status Tags work independent from the device connection. The state of the device connection is displayed on the Configuration page.

When handshake mode is used, the driver creates a directory under 'data/drivers' to store the message queues. When deleting a driver, this directories will not be deleted.

This driver does not support multiple subscription rates for a message. When there are multiple subscriptions to a message, the driver will update all subscriptions with the fastest rate.

If the device sends new messages *really* fast, the driver will start to discard single messages to prevent buffer overflows. There will be an error logged in this case. If the log is flooded with those messages, you should disable the driver and check the connected device.

Best practices

For logging batch data with a transaction group, it is recommended to use the delayed queue mode and the `_MessageCount` tag as a trigger.
To prevent logging of null values on group start, a 'run always' expression tag may be used:
`hasChanged({[~]_MessageCount}) && coalesce({[~]_MessageCount},0)>0`
Use this tag as a group trigger with trigger condition 'is > 0'.
Please note that this will only work in subscribed OPC mode, so make sure that there are no other subscriptions to tags used in the group, as this may result in inconsistent data.

The delayed queue mode is quite simple to use, but there are some things that affect transaction groups:
- Make sure to set the transaction groups rate faster than 3 times the rate data arrives from the device.
- Do not use any other subscriptions to _Timestamp or _MessageCount tags of this message (any subscription to this tags will activate publishing of that message).
- Pausing a transaction group does not cancel the subscription, so the driver will keep publishing data that is not handled by the paused group.

If using handshake or delayed queue mode, messages may be logged twice after a restart or a redundant failover. To prevent this, you may either use the _Timestamp tag as a unique key in the database (this will cause quarantined records in the store an froward system), or create a special insert trigger in the database that prevents insertion of double values.

Common error messages

- *Messages with IDxx arriving to fast. Discarded latest message.* The connected device sends messages faster than they can be evaluated or stored to the handshake queue. There's a maximum of 15 messages waiting for evaluation before this error is triggered.
- *Maximum queue size exceeded for message IDxx, discarding oldest message.* The maximum queue size is exceeded. Most probably messages are received at a faster rate than they were evaluated by the client. The maximum queue size is hardcoded to 500 entries per message.