DEWETRON

SOFTWARE REFERENCE MANUAL
OXYGEN ETHERNET RECEIVER XML CONFIGURATION

Version: 1.2
Date: 30/03/2020
Author: Markus Fauster

DEWETRON

The information contained in this document is subject to change without notice.

DEWETRON GmbH (DEWETRON) shall not be liable for any errors contained in this document. DEWETRON MAKES NO WARRANTIES OF ANY KIND WITH REGARD TO THIS DOCUMENT, WHETHER EXPRESS OR IMPLIED. DEWETRON SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

DEWETRON shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory, in connection with the furnishing of this document or the use of the information in this document.

Technical Support

Please contact your local authorized DEWETRON representative first for any support and service questions.

For Asia and Europe, please contact:

DEWETRON GmbH
Parkring 4
8074 Grambach
AUSTRIA
Tel.: +43 316 3070
Fax: +43 316 307090
Email: support@dewetron.com
Web: http://www.dewetron.com

For America, please contact:

DEWETRON, Inc.
PO Box 1460
Charlestown, RI 02813
U.S.A.
Tel.: +1 401 364 9464
Toll-free: +1 877 431 5166
Fax: +1 401 364 8565
Email: support@dewamerica.com
Web: http://www.dewamerica.com

The telephone hotline is available Monday to Friday between 08:00 and 17:00 GST (GMT -5:00)

Restricted Rights Legend:

Use Austrian law for duplication or disclosure.

DEWETRON GmbH
Parkring 4
8074 Grambach
AUSTRIA

Printing History:

Please refer to the page bottom for printing version.

DEWETRON

Before updating your software please contact DEWETRON. Use only original software from DEWETRON.

Please find further information at www.dewetron.com.

DEWETRON

DOCUMENT HISTORY

Document Type:	Technical Document
Document Name:	Ethernet Receiver XML
Document Owner:	Michael Oberhofer
Document ID:	\\path\Technical_Document.docx

Date	Author	Changes	Rev.
2018-11-19	M. Fauster	First revision of the document.	0.1
2018-11-21	M. Fauster	Added examples and xml schema	0.2
2018-12-04	M. Fauster	Description of valid_if attribute	0.3
		Version 1.0	1.0
2019-05-03	G. Neubauer	Description of synchronization node	1.1
2020-03-04	J. Pöllitsch	Added clarification to synchronization section	1.2

DEWETRON

TABLE OF CONTENTS

DEWETRON

1 BASIC STRUCTURE

An ethernet receiver instance has to be initialized using an xml file that describes the structure of the data packets, how they should be received and how the channels should be displayed.

A **`DataStream`** element describes how to receive and process a single packet stream. Multiple independent **`DataStream`**s can be defined in a single file to cover scenarios where a single remote system concurrently sends data using separate connections.

The file can also contain a single **`ChannelTopology`** which describes the order and grouping of the channels displayed in the channel list. If no topology is provided, all defined channels will be shown in the channel list in the same order as in the XML file.

More complex XML elements are described in the following format:

Description of the semantics of using the element.

<element name> Name of the element as used in the XML	***element***	***Parent: <element names>*** List of possible elements that can have this element as a child
<attribute name> Name of the attribute. Bold if the element is always mandatory.	<attribute type> or 'list', 'of', **'allowed'**, 'values' The default value may be highlighted in bold in a value list.	Detailed description of the attribute.

Receiver is the root element of the configuration file.

Receiver	***element***	***Parent: none***
name	string	This can be used to define the name of the created receiver instance in Oxygen. It is derived from the configuration file name if this attribute is omitted.

2 DATA STREAMS

A **`DataStream`** definition contains three major parts:

1. The source specifies where and how the bytes, sent by the remote system, are read from.
2. The **`PacketDefinition`** describes how to group these bytes into packets for parsing.
3. The configuration on how to transform bits and bytes from input packets into scaled samples for processing in Oxygen can be found in the **`Channels`** element.

DataStream	***element***	***Parent: Receiver***
name	string	Optional identifier of the data stream. No two streams in a single file may use the same name.

2.1 SOURCES

One of the source elements defined below has to be used to receive data in a stream.

2.1.1 UDP

`UDPSource` opens the specified UDP port and processes the payload of all incoming packets.
Since UDP is packet oriented already, it may be used without an additional PacketDefinition. All incoming packets are unconditionally parsed in this case.

UDPSource	***element***	***Parent: DataStream***
address	IPv4 address	Selects the network adapter to receive data from. Use '0.0.0.0' or omit the attribute to bind to all installed adapters.
Port	uint16	Specified on which port to listen for incoming packets.

`AcceptFrom` is an optional element that can limit which packets are accepted and added to the stream. Can be used multiple times in the same source to accept data from different hosts.

AcceptFrom	***element***	***Parent: UDPSource***
host	IPv4 address	IP source address of the packet.

2.2 PACKET DESCRIPTION

`PacketDefinition` defines how the single stream of bytes in segmented into packets for further processing. Additionally, it is possible to perform some basic validity checks in order to filter out packets that should not be parsed.

The following algorithm is used to extract packets:

- fixed length and nothing else: split stream into fixed size segments
- fixed bytes: search for byte sequence; if found subtract offset to find start of packet (all elements must match!)

Once the packet start is known, its length in bytes is determined by one of the following:

- length for fixed length
- the result of evaluating the **`VariableLength`** element and its embedded **`NumericValue`**.

PacketDefinition	***element***	***Parent: DataStream***
length	uint32	If set, packets are defined to have a fixed length in bytes. Incoming byte streams are always grouped into packets of this size.
max_length	uint32	If set, any discovered packets are truncated to this size.

All **`FixedPacketByte`**s must match in order to successfully detect a valid packet. This element can be used to filter based on a protocol header or a specific packet type field.

FixedPacketByte	***element***	***Parent: PacketDefinition***
offset	uint32	Offset of the byte within the packet (0 is the first byte)
value	uint8	Expected value of the byte (use prefix 0x for hex values)

Using the **`VariableLength`** element it is possible to correctly detect packets that contain an embedded length identifier. This element contains a **`NumericValue`** element (optionally followed by scaling) as a child which is used to calculate that value

2.3 SYNCHRONIZATION DESCRIPTION

In case of Synchronization a **`DataStream`** needs to contain a single Synchronization element, which contains definition and decoding information the timestamping information. Timestamping information needs to be a sample within the stream packets, defined within an inferior element described below. Additionally, such a sample might be also decoded as a distinct **`Channel`**, as shown in the examples. If Synchronization is used, the received timestamping information determines the position of the samples on the oxygen acquisition timeline.
Note, that a given timestamp is only accepted as valid if it within a range of 10 seconds around Oxygens current acquisition time and if it is advanced to any timestamp processed prior.
Keep in mind: The oxygen acquisition time is not necessarily in sync with the operating system time.
Therefore, synchronization of operating system times does **not** imply that sample data is synchronized over multiple oxygen systems. Especially in longer running measurements, the oxygen acquisition time of multiple systems will drift apart if there is no oxygen synchronization (e.g. TRION sync, IRIG, ...) established.

2.3.1 RELATIVETIMESTAMPCHANNEL

A **`RelativeTimestampChannel`** holds the sample information for a timestamping method which is based relatively to a fixed time of day. It is needed to contain a sub-item defining a **`Sample`** which holds the actual decoding information.

RelativeTimestampChannel	*element*	*Parent: Synchronization*
base	string	Definition of time of day where the channel is based relatively. This item is required and has to be equal to one of the following pre-defined values: midnight
unit	string	Defines the time unit of the timestamp information. This item is required and has to be equal to one of the following pre-defined values: ms
offset	int32	It has to be noted that **`Oxygen`** timestamping is based on absolute UTC timestamps. Therefore, a correction value might be applied here to fix clock deviations like time-zone issues or fixed delays and offsets.

2.4 CHANNEL DESCRIPTION

A **`DataStream`** contains a single **`Channels`** element which contains decoding information for extracting its channels from a data packet. In addition, it is possible to define **`SelectorValue`**s that may be used for conditional decoding of channel, for example depending on a packet type identifier.

2.4.1 CHANNELS

Channel	*element*	*Parent: Channels*
name	string	Name that is used to identify the channel. It has to be unique within a single configuration file.
unit	string	The unit of the physical value that is measured by the channel. SI units and prefixes should be used if possible.
description	String	An optional textual description of the channel and/or the interpretation of its values.
type	‘double’, ‘utf8’	Describes the format of the channel that will be generated in oxygen.

		Double: creates a scalar channel that contains numeric samples. These values are parsed and scale according to the rules specified in the **`NumericValue`** element before being written to the channel. UTF-8 identifies a channel where each sample is a Unicode string of arbitrary length. A **`ByteBlockValue`** specifies the relevant byte range which is then interpreted as a string.

Sample specifies the processing steps that generate the final channel value.

Sample		***Parent: Channel, RelativeTimestampChannel***
valid_if	string (expression)	Can contain a logical expression which is evaluated to determine if the sample should be read for a specific packet. If omitted or left empty it is always considered true. Each condition has the following format: <SelectorName> = <Value> e.g. 'type = 4' Multiple conditions can be combined using the && (logical AND) and || (logical OR) operators like this: 'type=4 || type=5"

Numeric value describes how to derive a floating-point channel sample from a number of input bits in the packet. The offsets and length attributes select the range of bits that are to be processed. These bits are then interpreted depending on the selected type and the ordering attributes to get the unscaled raw number. This number is then scaled to the final value based on the scaling parameters provided as children.

NumericValue		***Parent: Sample, SelectorValue, VariableLength***
byte_offset	uint32	Offset of the first interesting byte in the packet.
bit_offset	uint8	Offset, relative to byte_offset, of the first interesting bit.
bit_length	uint8	Number of interesting bits.
type	'unsigned', 'signed', 'float'	Specifies how the input bits are interpreted as a numeric value: Unsigned: bit are in a sequence of arbitrary length and interpreted positionally depending of the order attributes. Signed: an unsigned value is parsed and interpreted as a 2-complement signed value.

		Float: interpreted as an IEEE floating point value. This type only supports values of 32 and 64 bits length.
byte_order	'msb_first', **'msb_last'**	Selects the interpretation of bytes if bit_length > 8.
bit_order	'msb_0', **'lsb_0'**	Selects the interpretation of bit offsets and bit values.

A NumericValue can be followed by a **`LinearScaling`** element that converts the parsed raw numeric value to the final value that is handed over to oxygen. It uses the formula final_value = raw_value * scale + offset.

LinearScaling	***element***	***Parent: Sample***
scale	double	Default value is 1.
offset	double	Default value is 0.

Using the **ByteBlockValue** element it is possible to extract a byte sequence of fixed length from the input packet. This sequence can be interpreted as string or an NMEA sentence (see Channel.type) and stored in a matching channel.

ByteBlockValue	***element***	***Parent: Sample***
byte_offset	uint32	Offset of the first byte within the packet
byte_length	uint32	Number of bytes to use, starting from the first byte

Setting a display range for a numeric channel helps when using it in Oxygen. The default range of certain instruments is influenced by these settings for example.

DisplayRange	***element***	***Parent: Channel***
min	double	Lower bound of the expected output range of the channel.
max	double	Upper bound of the expected output range of the channel.
resolution	uint8	Indicates the number of significant digits (of the scaled value) after the decimal point. This value is currently not used in Oxygen.

2.4.2 SELECTORS

`SelectorValue`s can be defined in the **`Channels`** element and referenced in **`Channel`**/**`Sample`**.valid_if if conditional decoding is required. It contains a **`NumericValue`** element which describes how the selectors actual value is extracted from the packet. valid_if conditions are true if the the compared values are equal.

SelectorValue	***element***	***Parent: Channels***
name	string	Name of the selector value.

DEWETRON

3 TOPOLOGY

Using the **`ChannelTopology`** element it is possible to specify a logical grouping of the parsed channels and make working with the remote system easier.
If no topology is provided, all channels are shown without further structure in all Oxygen channel lists.

Creates a channel group within the parent group or topology.

ChannelGroup	***element***	***Parent: ChannelTopology, ChannelGroup***
name	string	Group name as shown in the channel list.

Shows a channel in the parent group or topology.

ChannelRef	***element***	***Parent: ChannelTopology, ChannelGroup***
channel_name	string	Name of the channel that should be shown.

DEWETRON

4 EXAMPLES

4.1 SIMPLE CONFIGURATION

4.1.1 DATA

'D'	*'T'*	*Counter [uint32]*	*AI1 [sint16]*

- UDP packets with 8 bytes payload are sent to port 1021
- 2 byte header: 'DT'
- 4 byte unsigned counter value
- 2 byte signed integer analog channel; measurement range is ±10V

4.1.2 CONFIGURATION

```
<?xml version="1.0"?>
<Receiver xmlns="http://xml.dewetron.com/receiver">
 <DataStream>
  <!-- Receive all packets on UDP port 1021 -->
  <UDPSource port="1021" />
  <!-- All packets are 8 bytes long and the first two bytes are fixed -->
  <PacketDefinition length="8">
   <FixedPacketByte offset="0" value="0x44" />
   <FixedPacketByte offset="1" value="0x54" />
  </PacketDefinition>
  <Channels>
   <!-- First channel is a 32 bit counter -->
   <Channel name="Counter" type="double">
    <Sample>
     <NumericValue byte_offset="2" bit_length="32" type="unsigned" />
    </Sample>
   </Channel>
   <!-- Second channel is the 16bit raw value of an analog input -->
   <Channel name="AI1" unit="V" type="double">
    <DisplayRange min="-10" max="+10" resolution="3" />
    <Sample>
     <NumericValue byte_offset="6" bit_length="16" type="signed" />
     <LinearScaling scale="0.00030517578125" offset="0" />
    </Sample>
   </Channel>
  </Channels>
 </DataStream>
</Receiver>
```

4.2 Conditional Decoding

4.2.1 Data

'P'	*[1, 2, 3]*	*Coordinate [float]*

- UDP packets with 6 bytes payload are sent to port 1021
- 1 byte header: 'P'
- 1 byte describes which coordinate is sent in this packet (1=>X; 2=>Y; 3=>Z)
- 4 byte float value of the specified coordinate

4.2.2 Configuration

```
<?xml version="1.0"?>
<Receiver>
 <DataStream>
  <UDPSource port="1021" />
  <PacketDefinition length="6">
   <FixedPacketByte offset="0" value="0x50"/>
  </PacketDefinition>
  <Channels>
   <SelectorValue name="which">
     <NumericValue byte_offset="1" bit_length="8" type="unsigned"/>
   </SelectorValue>
   <Channel name="X" type="double">
    <Sample valid_if="which=1">
     <NumericValue byte_offset="2" bit_length="32" type="float"/>
    </Sample>
   </Channel>
   <Channel name="Y" type="double">
    <Sample valid_if="which=2">
     <NumericValue byte_offset="2" bit_length="32" type="float"/>
    </Sample>
   </Channel>
   <Channel name="Z" type="double">
    <Sample valid_if="which=3">
     <NumericValue byte_offset="2" bit_length="32" type="float"/>
    </Sample>
   </Channel>
  </Channels>
 </DataStream>
 <ChannelTopology>
  <ChannelGroup name="Location">
   <ChannelRef channel_name="X"/>
   <ChannelRef channel_name="Y"/>
   <ChannelRef channel_name="Z"/>
  </ChannelGroup>
 </ChannelTopology>
</Receiver>
```

4.3 SYNCHRONIZATION

4.3.1 DATA

Timestamp information [uint64]

- UDP packets with 8 bytes payload are sent to port 1021
- 8 byte timestamp information, as milliseconds relative to midnight, with a 2 hour offset (UTC+2), decoded as timestamp and displayed as channel

4.3.2 CONFIGURATION

```
<?xml version="1.0"?>
<Receiver>
 <DataStream name="Time_Stream">
  <UDPSource port="1021"/>
  <PacketDefinition length="8" />
  <Synchronization>
   <!-- Timestamping relativ to midnight (ms from midnight with 2hrs offset) -->
   <RelativeTimestampChannel base="midnight" unit="ms" offset="7200000" >
    <Sample>
     <NumericValue byte_offset="0" bit_offset="0" bit_length="64" type="unsigned" />
    </Sample>
   </RelativeTimestampChannel>
  </Synchronization>
  <Channels>
   <!-- Decode timestamping information as "Time" channel -->
   <Channel name="Time" unit="ms" type="double">
    <Sample>
     <NumericValue byte_offset="0" bit_offset="0" bit_length="64" type="unsigned" />
    </Sample>
    <DisplayRange min="0" max="+86400000" resolution="1" />
   </Channel>
  </Channels>
 </DataStream>
</Receiver>
```

5 SCHEMA

```
# Oxygen Ethernet Receiver Configuration
#
# RelaxNG compact syntax specification (http://www.relaxng.org/compact-tutorial-20030326.html)
#
# XSD schema can be generated from on this file using the free Trang schema converter:
# java -jar trang.jar -I rnc -O xsd receiver.rnc receiver_generated.xsd

default namespace = "http://xml.dewetron.com/receiver"

grammar
{

    boolean = "true" | "false"

    anyElement =
      element * {
        (attribute * { text }
         | text
         | anyElement)*
      }
    anyContent = mixed { anyElement* }
    ToDo = anyElement*

    unsigned_hex_or_dec = xsd:integer | xsd:string { pattern='0[xX][0-9a-fA-F]+' }

  accept_from = element AcceptFrom
  {
    attribute host { text }
  }

  udp_source = element UDPSource
  {
    attribute address { text }? #ip of the network device to listen on
    &
    attribute port { unsigned_hex_or_dec } #udp port
    &
    accept_from*
  }

  source = udp_source

  display_range = element DisplayRange
  {
    attribute min { xsd:double } #default range for instruments
    &
    attribute max { xsd:double } #default range for instruments
    &
    attribute resolution { xsd:integer }? #number of decimal places that should be shown
  }

  numeric_value = element NumericValue
  {
    attribute byte_offset { unsigned_hex_or_dec }?
    &
    attribute bit_offset { unsigned_hex_or_dec }?
    &
    attribute bit_length { unsigned_hex_or_dec }
    &
    attribute type { "unsigned" | "signed" | "float" }
    &
    attribute byte_order { "msb_first" | "lsb_first" }?
    &
```

```
  attribute bit_order { "msb_0" | "lsb_0" }?
}

block_terminator = element BlockTerminator
{
  attribute alignment { text }?
  &
  attribute value { text }
}

byte_block_value = element ByteBlockValue
{
  attribute byte_offset { unsigned_hex_or_dec }
  &
  attribute byte_length { unsigned_hex_or_dec }
}

scaling  = element LinearScaling
{
  attribute scale { xsd:double }
  &
  attribute offset { xsd:double }
}

numeric_chain = numeric_value , scaling?

sample = element Sample
{
  attribute valid_if { text }?
  &
  (
    numeric_chain
    |
    ( byte_block_value )
  )
}

channel = element Channel
{
  attribute name { text }
  &
  attribute short_name { text }?
  &
  attribute type { "double" | "nmea" | "utf8" }
  &
  attribute unit { text }?
  &
  attribute description { text }?
  &
  display_range?
  &
  sample
}

selector_value = element SelectorValue
{
  attribute name { text }
  ,
  numeric_chain
}

variable_length = element VariableLength
{
  numeric_chain
}

fixed_packet_byte = element FixedPacketByte
{
```

```
  attribute offset { unsigned_hex_or_dec }
  &
  attribute value { unsigned_hex_or_dec }
}

packet_definition = element PacketDefinition
{
  attribute length { xsd:unsignedInt }?
  &
  attribute max_length { xsd:unsignedInt }?
  &
  variable_length?
  &
  fixed_packet_byte*
}

relative_timestamp_channel = element RelativeTimestampChannel
{
  attribute base { "midnight" }
  &
  attribute unit { "ms" }
  &
  attribute offset { xsd:unsignedInt }?
  &
  sample
}

synchronization = element Synchronization
{
  relative_timestamp_channel
}

data_stream = element DataStream
{
  source
  ,
  packet_definition?
  ,
  element Channels {
    channel*
    &
    selector_value*
  }
}

channel_ref = element ChannelRef
{
  attribute channel_name { text }
}

channel_group = element ChannelGroup
{
  attribute name { text }
  &
  channel_ref*
  &
  channel_group*
}

channel_topology = element ChannelTopology
{
  channel_group+
}

# root element
start = element Receiver
{
  data_stream+
```

```
    ,
    channel_topology?
  }

}
```