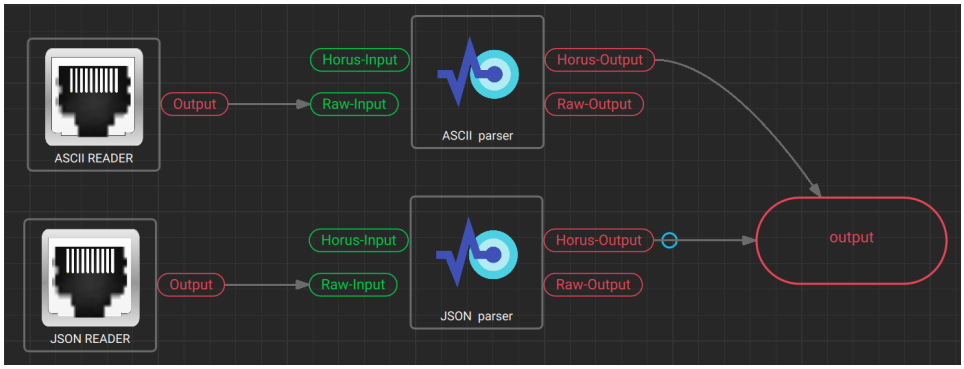
HORUS VIEW & EXPLORE

TECHNICAL DOCUMENTION

ASCII/JSON Sensor Protocol

Horus Embed
embed@horus.nu

Horus View & Explore
info@horus.nu



June 18, 2020

Contents

1 Introduction: ASCII / JSON Sensor Protocol

This relatively small document describes two different methods of generating sensor-messages using Horus View & Explore's system v2. The **ASCII** protocol describes a state machine approach while the **JSON** is a 'complete' message oriented approach. What this document is not: an overview on how sensor messages can be viewed and used to manipulate components that are currently active.

2 Horus View & Explore ASCII Sensor Protocol

2.1 Introduction: ASCII Sensor Protocol

The Sensor protocol is an easy protocol to understand, remember and implement. Its strength is therfor shown during implementation and as commandline interface. Since it is implemented as an state machine only changes in value, name and structure need to be addressed.

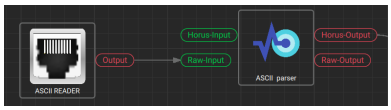
The following listing shows a series of easy to understand commands

```
NAME temprature VALUE 0.1
VALUE 0.2
VALUE 0.3
STRUCTURE VECTOR
DIM_START 3 DIM_STOP
VALUES 1 2 3
STRUCTURE MATRIX
DIM_START 2 3 DIM_STOP
VALUES 1 2 3 4 5 6
```

2.2 In practice: connectivity

There are several ways to insert sensor messages within the Horus environment. The one that we will focus on is network connectivity.

2.3 Network connectivity



Connecting a **Network Reader** { Protocol: TCP, Behavior: Read } to a **Sensor parser** {incomming protocol: ASCII} allows for ASCII/Sensor over Network communication.

Using simple commandline utilities such as netcat allows messages to be send to the Network Reader.

```
$>> nc -4 localhost 5567
NAME Pressure VALUE 3
VALUE 3.5
VALUE 3
```

2.4 The protocol parser.

As we have seen in the connectivity section, the ASCII protocol parser is easily embedded within a system v2 system. The ASCII protocol parser is implemented as a state-machine, meaning that the following statements will not generate a sensor message:

```
STRUCTURE VECTOR
DIM_START 3 DIM_STOP
STRUCTURE VECTOR
DIM_START 3 DIM_STOP
```

nor do the last two commands change the state of the parser.
It does make sense however to start your session by providing:

```
NAME mySensor STRUCTURE VECTOR DIM_START 3 DIM_STOP DATATYPE UINT
```

after which all generated sensor message will be vectors of unsigned integers with length 3. After which commands such as:

```
VALUE 1 2 3
VALUES 3 4 5
VALUES 0 1 2
```

can be issued. Resulting in the following vectors: $\begin{bmatrix}1 & 2 & 3\end{bmatrix}$, $\begin{bmatrix}3 & 4 & 5\end{bmatrix}$, $\begin{bmatrix}0 & 1 & 2\end{bmatrix}$.

2.5 The protocol definition.

TOKEN	ARGUMENT(S)
NAME	literal with no spaces
UNITS	DEGREES, RADIANS, NORMALIZED, UNKNOWN_UNITS
STRUCTURE	SCALER, VECTOR, MATRIX, COMPLEX
DATATYPE	STRING, INT, DOUBLE, FLOAT, LONG, UINT, ULONG
VALUE	list of values, that corresponds to the dimensions.
VALUES	list of values, that corresponds to the dimensions.
DIM_START	list of values, where each entry corresponds to one dimension.
DIM_STOP	
RESET	reset parser in default state.

DEFAULT STATE TOKEN	 ARGUMENT(S)
NAME	unknown
UNITS	UNKNOWN_UNITS
STRUCTURE	SCALER
DATATYPE	DOUBLE

3 Horus View & Explore JSON Sensor message

3.1 Introduction: JSON Sensor Message

The JSON sensor message is a short message describing the properties of a sensor value.
The following datastructure is an example of a sensor message.

```
{
  "Name": "temp",
  "Structure": 1,
  "Data": 3,
  "DoubleValue": [
    33.0
  ],
  "Dimension": [
    1
  ]
}
```

3.2 In practice: connectivity

There are several ways to insert sensor messages within the Horus environment. The one that we will focus on is network connectivity.

3.3 Network connectivity

Connecting a **Network Reader** { Protocol: TCP, Behavior: Read } to a **Sensor parser** {incomming protocol: Horus-JSON} allows for JSON/Sensor over Network communication.

Using simple commandline utilities such as netcat allows messages to be send to the Network Reader.

```
$>> nc -4 localhost 5567
{"Name": "temp", "Structure": 1, "Data": 3,  ...
   .. "DoubleValue": [33.0], "Dimension": [1]}
```

When sending messages always remove newlines!

3.4 Message definition.

Datatype definition		
Type	Data	name–value pair
UNKNOWN	0	
STRING	1	StrValue
INTEGER	2	IntValue
DOUBLE	3	DoubleValue
FLOAT	4	FloatValue
LONG	5	LongValue
UNSIGNED_INTEGER	6	UnsignedIntValue
UNSIGNED_LONG	7	UnsignedLongValue

Structure definition	
Type	Structure
UNKNOWN	0
SCALAR	1
VECTOR	2
MATRIX	3
COMPLEX	4

Dimension definition	
Type	Dimension
SCALAR	[1]
VECTOR	[length]
MATRIX	[rows, cols]
COMPLEX	[dim_0 length, .. , dim_n length]

3.5 Examples

3.5.1 Matrix of integers

```
{
  "Name": "temp",
  "Units": {
   "UnitValue": 2
  },
  "Structure": 3,
  "Data": 2,
  "IntValue": [
    1,
    2,
    3,
    4,
    5,
    6
  ],
  "Dimension": [
    3,
    2
  ]
}
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Units definition (optional)			
Unit	Value	Unit	Value
Unknown	0	Kilometers per hour	40
Unknown	1	Beats per minute	41
Degree	2	Pixel	42
Radian	3	Microsesconds	50
Celcius	4	kilogram	51
Normalized	5	milliBar	52
Degree_sec	22	rotations per minute	53
Radian_sec	23	megahertz	54
g-force	24	gigabyte	55
Gauss	25	Ampere hour	56
Percentage	26	Pascal	57
Volts	27	kilopascal	58
Wats	28	grams/sec	59
Amps	29	Fuel-air equivalence ratio	60
Bit	30	Liters per hour	61
Quaternion	31	Seconds	62
Millimeters	32	Minutes	62
Meters	33		
Kilometers	34		
Meters per second	39		

3.5.2 Vector of doubles

```
{
  "Name": "temp",
  "Units": {
   "UnitValue": 2
  },
  "Structure": 2,
  "Data": 3,
  "DoubleValue": [
    1.1,
    2.2,
    3.3
  ],
  "Dimension": [
    3
  ]
}
```

$$\begin{bmatrix} 1.1 \\ 2.2 \\ 3.3 \end{bmatrix}$$

3.5.3 Scalar of unsigned integers

```
{
  "Name": "mySensor",
  "Structure": 1,
  "Data": 6,
  "UnsignedIntValue": [
    7
  ],
  "Dimension": [
    1
  ]
}
```

$$7$$