## Contents

1 Description 1

2 20 Program Organization Units 3
   2.1 Data types 3
      2.1.1 typParameters (STRUCT) 3
      2.1.2 typSentence (STRUCT) 3
   2.2 Enumeration 3
      2.2.1 eSentenceType (ENUM) 3
   2.3 FbNmeaSubscriber (FB) 3
      2.3.1 FbNmeaSubscriber.GetEachSentence (METH) 8
      2.3.2 FbNmeaSubscriber.GetFilteredSentence (METH) 9
      2.3.3 FbNmeaSubscriber.WriteSentence (METH) 11

3 80 Status 13
   3.1 ErrorNmea (GVL) 13
   3.2 eErrorNmea (ENUM) 14

4 Parameter (PARAMS) 15

5 VersionHistory (GVL) 16

6 Library Reference 17
CHAPTER 1

Description

This document is automatically generated. Because of this, the chapter 30 Visualization is not shown in this document. If you are interested in getting to know more about visualization, we refer to the library manager of e!Cockpit.

Subject to Changes

WAGO Kontakttechnik GmbH & Co. KG reserves the right to provide for any alterations or modifications. WAGO Kontakttechnik GmbH & Co. KG owns all rights arising from the granting of patents or from the legal protection of utility patents. Third-party products are always mentioned without any reference to patent rights. Thus, the existence of such rights cannot be excluded.

Personnel Qualification

All tasks that are carried out with libraries made for the e!COCKPIT software must only be performed by qualified electrical specialists instructed in PLC programming according to IEC 61131-3.

All tasks that have an effect on the properties or the behavior of automation hardware or software products must only be performed by qualified employees with a thorough knowledge of handling the products concerned.

Intended Use of e!COCKPIT Libraries

Libraries created for the e!COCKPIT software are used to simplify the development of application projects in the IEC 61131-3 programming languages.

For automation tasks, WAGO offers programmable logic controllers in a wide variety of performance classes. In combination with a wide range of I/O modules, the controllers can process standard types of field signals. Controllers can be implemented centrally or in decentralized configurations. The controllers offer interfaces for the most commonly used fieldbuses for use in decentralized configurations. Fieldbus independent I/O modules are then linked via fieldbus couplers. WAGO controllers offer a runtime environment for user programs called e!RUNTIME. Software projects for implementation in e!RUNTIME environments can be created in e!COCKPIT. The programming environment in e!COCKPIT is based on the established CODESYS 3 industrial standard. Users with a previous knowledge of CODESYS 3 will thus find this environment largely familiar. The following programming languages of the IEC 61131-3 standard are available:

- Structured Text (ST)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
- Instruction List (IL)
- Sequential Function Chart (SFC)
- Continuous Function Chart (CFC)

The individual programming languages can also be combined as required during the development of the software. A portfolio of prepared libraries can be accessed for many frequently used functions in order to make software development more efficient. This document provides an overview of the WagoAppSerial_NMEA that WAGO offers for e!COCKPIT.
Functionblocks for serial NMEA 0183 communication

Further library information are summerized here:

Company  WAGO
Title    WagoAppSerial_NMEA
Version  1.6.2.0
Categories WAGO LayerView/App; Application; WAGO FunctionalView/Connectivity/Serial
Namespace WagoAppSerial_NMEA
Author   WAGO / u010545
Placeholder WagoAppSerial_NMEA

---

The content of this file was automatically generated with None on 13.08.2019, 20:55:07
20 Program Organization Units

2.1 Data types

2.1.1 typParameters (STRUCT)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>BYTE</td>
<td>quantity of values</td>
</tr>
<tr>
<td>Value</td>
<td>ARRAY [1..NMEA_MAX_VALUES_PER_SENTENCE] OF STRING(NMEA_MAX_VALUE_LENGTH)</td>
<td></td>
</tr>
</tbody>
</table>

2.1.2 typSentence (STRUCT)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>eSentenceType</td>
<td>eSentenceType</td>
<td>Type of this sentence -&gt; parametric / encapsulation / ...</td>
</tr>
<tr>
<td>sAddressField</td>
<td>STRING(5)</td>
<td></td>
</tr>
<tr>
<td>utParameter</td>
<td>typParameters</td>
<td></td>
</tr>
<tr>
<td>sRawSentence</td>
<td>STRING</td>
<td></td>
</tr>
</tbody>
</table>

2.2 Enumeration

2.2.1 eSentenceType (ENUM)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMEA_UNKNOWN_TYPE</td>
<td>0</td>
</tr>
<tr>
<td>NMEA_PARAMETRIC_SENTENCE</td>
<td>16#1</td>
</tr>
<tr>
<td>NMEA_ENCAPSULATION_SENTENCE</td>
<td>16#2</td>
</tr>
<tr>
<td>NMEA_PROPRIETARY_SENTENCE</td>
<td>16#4</td>
</tr>
<tr>
<td>NMEA_ALL_SENTENCES</td>
<td>16#FF</td>
</tr>
</tbody>
</table>

2.3 FbNmeaSubscriber (FB)

Interface variables
WagoAppSerial_NMEA, Release 1.6.2.0

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>xConnect</td>
<td>BOOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I_Port</td>
<td>WagoType-sCom.I_WagoSysComBase</td>
<td></td>
<td>Name of the Interface (e.g. ‘COM1’, ‘SER7.2’, )</td>
</tr>
<tr>
<td></td>
<td>ePhysical</td>
<td>WagoType-sCom.eTTYPhysicalLayer</td>
<td></td>
<td>RS232, RS422, RS485, etc</td>
</tr>
<tr>
<td></td>
<td>xAcceptIncomingWithoutCrc</td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE means accept frames without *crc but if a crc is present it have to be correct</td>
</tr>
</tbody>
</table>

| Output| xIsConnected                | BOOL             |              |                                                                         |
|       | xError                      | BOOL             |              |                                                                         |
|       | oStatus                     | WagoSysError-Base.FbResult |              |                                                                         |
|       | xNewSentenceReceived        | BOOL             |              | impulse for one cycle if a new sentence is received                    |
|       | utRxSentence                | typSentence      |              | last valid received sentence                                            |

**Function**

NMEA - Receiver for receiving all NMEA-Sentences and send all given sentences. This FB contains the basic functionality for sending and receiving NMEA sentences. You have to call this function block once for each cycle in your project to process the common functionality.

To get sentences from all the internal received sentences this FB provides two methods (see `FbNmeaSubscriber.GetFilteredSentence` and `FbNmeaSubscriber.GetEachSentence`)

For sending any sentence it will supported by the method `FbNmeaSubscriber.WriteSentence`

**Note:** You should always call this FB cyclic.

**Graphical Illustration**

![Graphical Interface of FbNmeaSubscriber](image)

**Example**

For receive and write sentences. If you only want to receive sentences you may delete the part for sending (`WriteSentence(..)`). Also you may delete the part for get sentence if your want to send only.

```plaintext
VAR
  //... common object ............................
  myNmeaSubscriber : FbNmeaSubscriber;

  //... write objects ............................
  myWriteSentence : typSentence := ( 
      SentenceType := eSentenceType.NMEA PARAMETRIC_SENTENCE,
      AddressField := 'AABCD',
      Parameter := (Count := 2, Value := ['123', 'abc'])
    );
```

2.3. FbNmeaSubscriber (FB)
my_oStatusWrite : WagoSysErrorBase.FbResult;
xWrite : BOOL;
xReset : BOOL;

//... get filtered sentences .............
myFilteredSentence : typSentence;

//... get each sentence .................
myEachSentence : typSentence;
END_VAR

// COMMON PART : (ALWAYS NEEDED)
//===============================================================================
// This is the basic part of a nmea subscriber
// This call is need to receive all sentences from the serial line
// and for send all sentences given by "WriteSentence" to the
// serial line. You need this call once for each cycle in your project.
//===============================================================================
myNmeaSubscriber(
  xConnect := TRUE, // open the serial port
  I_Port := my652, // place here the name of used serial port
  ePhysical := WagoTypesCom.eTTYPhysicalLayer.RS232,
  xIsConnected => , // get here an information about successfully
  →
  →
  → open the serial port
  xError => , // get here an information about any error
  →
  →
  → oStatus => , // get here additional Information if an error
  →
  → ocorred
  →
  → xNewSentenceReceived => , // impulse for one cycle if a new sentence is
  →
  → received
  →
  → utRxSentence => , // last valid received sentence
  →
  → xAcceptIncomingWithoutCrc := FALSE // if you place here TRUE sentences
  →
  → without crc will be accepted
);
//
//=================================================================================
// WRITE PART : [optional]
//=== WRITE A SENTENCE TO SERIAL LINE ====================================== IF xWrite THEN // set xWrite for once at TRUE to write a sentence

  myNmeaSubscriber.WriteSentence( xEnable := xWrite,
  xWithoutCrc := FALSE,
  utSentence := myWriteSentence,
  oStatus := my_oStatusWrite,
  xReset => xReset
  );

  xWrite R= xReset; // reset this variable when the sentence is written to the
  →
  → pipe
  →
  → // or an error occured

  IF my_oStatusWrite.IsError() THEN // you have to process the error
    // process here any errors
  END_IF

END_IF
//-------------------------------------------------------------------------------

// GET EACH SENTENCE PART : [optional]
//=== GET EACH SENTENCE ==============================================================
// If you want to get all sentences received by NmeaReceiver
// then you can use this method
//-------------------------------------------------------------------------------

2.3. FbNmeaSubscriber (FB)
IF myNmeaSubscriber.GetEachSentence(utSentence:= myEachSentence) THEN

    // you stay here once when a sentence was received
    // process here the new 'myEachSentence'

END_IF

//========================================================================================

// GET FILTERED SENTENCE PART: [optional]
//= GET A FILTERED SENTENCE
//=--------------------------------------------------------------------------------------------
//= If you want to get a filtered sentence from all sentences received by NmeaReceiver
//= then you can use this method to get the only one you have specified by the sentence type
//= and the address pattern.
//= For the sentence type you may combine with 'OR' different types
//= At the address pattern you may use wildcards '*' and '?'
//=-------------------------------------------------------------------------------------------

IF myNmeaSubscriber.GetFilteredSentence(
    eSentence_Type := eSentenceType.NMEA_PARAMETRIC_ SENTENCE
    OR eSentenceType.NMEA_PROPRIETARY_,
    sAddressPattern := 'PP*',
    xCaseSensitve := FALSE,
    utSentence := myFilteredSentence
) THEN

    // you stay here once when one of your filtered sentences was received
    // process here the new 'myFilteredSentence'

END_IF

//==========================================================================================

Example for FUP

2.3. FbNmeaSubscriber (FB)
// COMMON PART: (ALWAYS NEEDED)

// This is the basic part of a nmea subscriber
// This call is need to receive all sentences from the serial line
// and for send all sentences given by "WriteSentence" to the
// serial line.

myNmeaSubscriber

myNmeaSubscriber.WriteSentence

myWriteSentence

myStatusWrite

myFbNmeaSubscriber.GetEachSentence

myEachSentence

// WRITE PART: (optional)

// If you set Enable to TRUE then your sentence will be written to a pipe
// forprocessed and transmit by the basic functionality. Enable should reseted
// by the output xReset. After this you have to check the status object for errors.

myFbNmeaSubscriber.WriteSentence

myFbNmeaSubscriber.GetEachSentence

// GET EACH SENTENCE PART: (optional)

// If you do not need each sentences you can use the more powerful method
// GetFilteredSentence() to filter the sentences for your need.
// See the example at the next network.
// At the right output side you get an impulse for each received sentence.

// GET FILTERED SENTENCE PART: (optional)

// If you want to get a filtered sentence from all sentences received by NmeaReceiver
// then you can use this method to get the only one you have specified by the sentence type
// and the address pattern.
// For the sentence type you may combine with 'OR' different types
// At the address pattern you may use wildcards '*' and '?'

myFbNmeaSubscriber.GetFilteredSentence

2.3. FbNmeaSubscriber (FB)
2.3.1 FbNmeaSubscriber.GetEachSentence (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetEachSentence</td>
<td>BOOL</td>
<td></td>
</tr>
<tr>
<td>Inout</td>
<td>utSentence</td>
<td>typSentence</td>
<td>last valid, received and filtered sentence</td>
</tr>
</tbody>
</table>

Function Get each received sentences.

Note: For use this method you have to call the FbNmeaSubscriber cyclic.

Graphical Illustration

![Graphical Interface of FbNmeaSubscriber.GetEachSentence](image)

Example For receive each sentences.

```c
VAR
    //... common object ............................
    myNmeaSubscriber : FbNmeaSubscriber;
    
    //... get each sentence ................
    myEachSentence : typSentence;
END_VAR

// C O M M O N P A R T : (ALWAYS NEEDED)
//=================================================================================
// This is the basic part of a nmea subscriber
// This call is need to receive all sentences from the seriel line
// and for send all sentences given by "WriteSentence" to the
// serial line. You need this call once for each cycle in your project.
//=================================================================================
myNmeaSubscriber({
    xConnect := TRUE,                  // open the serial port
    I_Port := my652,                   // place here the name of used serial port
    ePhysical := WagoTypesCom.eTTYPhysicalLayer.RS232,
    xIsConnected => ,                  // get here an information about successfully
    xError => ,                        // get here an information about any error
    oStatus => ,                       // get here additional Information if an error
    xNewSentenceReceived => ,          // impulse for one cycle if a new sentence is
    utRxSentence => ,                  // last valid received sentence
    xAcceptIncomingWithoutCrc := FALSE  // if you place here TRUE sentences
    xwithout crc will be accepted
});
//=================================================================================
```

2.3. FbNmeaSubscriber (FB)
IF myNmeaSubscriber.GetEachSentence(utSentence:= myEachSentence) THEN
  // you stay here once when a sentence was received
  // process here the new 'myEachSentence'
END_IF

2.3.2 FbNmeaSubscriber.GetFilteredSentence (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetFiltered-Sentence</td>
<td>BOOL</td>
<td></td>
<td>set wanted sentence-type if set to NMEA_PARAMETRIC_SENTENCE then get all</td>
</tr>
<tr>
<td>Input</td>
<td>eSentence_Type</td>
<td>eSentence-Type</td>
<td></td>
<td>set the filter by the addressfield '*' and '?' as wildcards are allowed -&gt; empty string means get all</td>
</tr>
<tr>
<td></td>
<td>sAddress-Pattern</td>
<td>STRING(5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xCaseSensitive</td>
<td>BOOL</td>
<td></td>
<td>FALSE means uppercase/lowercase will be ignored</td>
</tr>
<tr>
<td>Inout</td>
<td>utSentence</td>
<td>typSentence</td>
<td></td>
<td>last valid, received and filtered sentence</td>
</tr>
</tbody>
</table>

Function

Filters the sentences specified by sentence type and address pattern from all received sentences.
Sentence types may be combine by ‘OR’ to get different types. Address pattern may use wildcards ‘?’ and ‘*’.
- ‘?’ means one of any char
- ‘*’ means as many as you like of any char

Examples of pattern

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘AB*’</td>
<td>The address field have to start with ‘AB’ followed by as many of any chars as you like</td>
</tr>
<tr>
<td>‘<em>XY</em>’</td>
<td>The address field must contain anywhere ‘XY’</td>
</tr>
<tr>
<td>‘ABCDE’</td>
<td>The address field must exact match ‘ABCDE’</td>
</tr>
<tr>
<td>‘AB???’</td>
<td>The address field must have 5 characters and the first must be ‘AB’</td>
</tr>
</tbody>
</table>

Note: For use this method you have to call the FbNmeaSubscriber cyclic.

Graphical Illustration
Example  For receive filtered sentences.

VAR

//... common object .............................
myNmeaSubscriber : FbNmeaSubscriber;

//... get filtered sentences ............
myFilteredSentence : typSentence;
END_VAR

// C O M M O N P A R T : (ALWAYS NEEDED)
//===============================================================================
// This is the basic part of a nmea subscriber
// This call is need to receive all sentences from the serial line
// and for send all sentences given by "WriteSentence" to the
// serial line. You need this call once for each cycle in your project.
//===============================================================================
myNmeaSubscriber{
 xConnect := TRUE, // open the serial port
 I_Port := my652, // place here the name of used serial port
 ePhysical := WagoTypesCom.eTTYPhysicalLayer.RS232,
 xIsConnected => , // get here an information about successfully
 → open the serial port
 xError => , // get here an information about any error
 oStatus => , // get here additional Information if an error
 → occurred
 xNewSentenceReceived => , // impulse for one cycle if a new sentence is
 → received
 utRxSentence => , // last valid received sentence
 xAcceptIncomingWithoutCrc := FALSE // if you place here TRUE sentences
 → without crc will be accepted
};

// GET FILTERED SENTENCE PART : [optional]
//= GET A FILTERED SENTENCE
===============================================================================
// If you want to get a filtered sentence from all sentences received by
// NmeaReceiver
// then you can use this method to get the only one you have specified by the
// sentence type
// and the address pattern.
// For the sentence type you may combine with 'OR' different types
// At the address pattern you may use wildcards '*' and '?'
//=
IF myNmeaSubscriber.GetFilteredSentence(
2.3.3 FbNmeaSubscriber.WriteSentence (METH)

Interface variables

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>WriteSentence</td>
<td>BOOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input</td>
<td>xEnable</td>
<td>BOOL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xWithoutCrc</td>
<td>BOOL</td>
<td>FALSE</td>
<td>TRUE -&gt; send the sentences without crc</td>
</tr>
<tr>
<td>Inout</td>
<td>utSentence</td>
<td>typSentence</td>
<td></td>
<td>sentence to send</td>
</tr>
<tr>
<td></td>
<td>oStatus</td>
<td>WagoSysError-Base.FbResult</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>xReset</td>
<td>BOOL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Function  Write a NMEA sentence to the serial interface.

Note: For use this method you have to call the FbNmeaSubscriber cyclic.

Graphical Illustration

![Fig. 2.4: Graphical Interface of FbNmeaSubscriber.GetWriteSentence](image)

Example  For write a sentence.
VAR
   //... common object ............................
   myNmeaSubscriber : FbNmeaSubscriber;

   //... write objects ..........................
   myWriteSentence : typSentence := ( SentenceType := eSentenceType.NMEA_PARAMETRIC_SENTENCE,
                                       AddressField := 'AABCD',
                                       Parameter := (Count := 2, Value := ['123', 'abc']) );
   my_oStatusWrite : WagoSysErrorBase.FbResult;
   xWrite          : BOOL;
   xReset          : BOOL;
END_VAR

// C O M M O N P A R T : (ALWAYS NEEDED see FbNmeaSubscriber)
//===============================================================================
// This is the basic part of a nmea subscriber
// This call is need to receive all sentences from the serial line
// and for send all sentences given by "WriteSentence" to the
// serial line. You need this call once for each cycle in your project.
//===============================================================================
myNmeaSubscriber( xConnect := TRUE, // open the serial port
                   I_Port := my652, // place here the name of used serial port
                   ePhysical := WagoTypesCom.eTTYPhysicalLayer.RS232,
                   xIsConnected => , // get here an information about successfully
                                      // open the serial port
                   xError => , // get here an information about any error
                   oStatus => , // get here additional Information if an error
                   xNewSentenceReceived => , // impulse for one cycle if a new sentence is
                                      // received
                   utRxSentence => , // last valid received sentence
                   xAcceptIncomingWithoutCrc := FALSE // if you place here TRUE sentences
                   // without crc will be accepted
               );

//==========================================================================
2.3. FbNmeaSubscriber (FB)

//--- WRITE A SENTENCE TO SERIAL LINE -------------------------------------
IF xWrite THEN // set xWrite for once at TRUE to write a sentence
   myNmeaSubscriber.WriteSentence( xEnable := xWrite,
                                   xWithoutCrc := FALSE,
                                   utSentence := myWriteSentence,
                                   oStatus := my_oStatusWrite,
                                   xReset => xReset
                          );
   xWrite R= xReset; // reset this variable when the sentence is written to the
                    // pipe
   // or an error occured
   IF my_oStatusWrite.IsError() THEN // you have to process the error
      // process here any errors
   END_IF
END_IF

//---------------------------------------------------------------------------
### 3.1 ErrorNmea (GVL)

<table>
<thead>
<tr>
<th>Value</th>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eErrorNmea.OK</td>
<td>WagoTypesErrorBase.eSeverity.none</td>
<td>‘OK’</td>
</tr>
<tr>
<td>eErrorNmea.CRC_ERROR</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘CRC Error’</td>
</tr>
<tr>
<td>eErrorNmea.MISSING_CRC</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error missing crc’</td>
</tr>
<tr>
<td>eErrorNmea.INVALID_CHAR</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error invalid char’</td>
</tr>
<tr>
<td>eErrorNmea.SECOND_STARTCHAR</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error second startchar’</td>
</tr>
<tr>
<td>eErrorNmea.SECOND_CRC_DELIM</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error second crc delimiter’</td>
</tr>
<tr>
<td>eErrorNmea.CRC_DELIM_TO_EARLY</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error crc delimiter to early’</td>
</tr>
<tr>
<td>eErrorNmea.FIELD_DELIM_TO_EARLY</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error field delimiter to early’</td>
</tr>
<tr>
<td>eErrorNmea.HEX_DELIM_TO_EARLY</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error hex delimiter to early’</td>
</tr>
<tr>
<td>eErrorNmea.TO_MANY_PARAMETERS</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error to many parameters’</td>
</tr>
<tr>
<td>eErrorNmea.PARAMETER_TO_LONG</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error parameter to long’</td>
</tr>
<tr>
<td>eErrorNmea.INVALID_SENTENCE_TYPE</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error invalid sentence type’</td>
</tr>
<tr>
<td>eErrorNmea.INVALID_ADDRESSFIELD</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error invalid address field’</td>
</tr>
<tr>
<td>eErrorNmea.PIPE_INVALID_PARAMETER</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error Pipe invalid parameter’</td>
</tr>
<tr>
<td>eErrorNmea.PIPE_NO_DATA</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error Pipe no data available’</td>
</tr>
<tr>
<td>eErrorNmea.PIPE_FULL</td>
<td>WagoTypesErrorBase.eSeverity.error</td>
<td>‘Error Pipe no free space for new data -&gt; Pipe full’</td>
</tr>
</tbody>
</table>
## 3.2 eErrorNmea (ENUM)

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CRC_ERROR</td>
<td>16#100</td>
<td>calculated crc not EQ received crc</td>
</tr>
<tr>
<td>MISSING_CRC</td>
<td>16#200</td>
<td>Endchar before complete crc received</td>
</tr>
<tr>
<td>INVALID_CHAR</td>
<td>16#300</td>
<td>not expected char received</td>
</tr>
<tr>
<td>SECOND_STARTCHAR</td>
<td>16#400</td>
<td>second startchar (&quot;$&quot; or ‘!”) received</td>
</tr>
<tr>
<td>SECOND_CRC_DELIM</td>
<td>16#500</td>
<td>second crc-delimiter ( * ) received</td>
</tr>
<tr>
<td>CRC_DELIM_TO_EARLY</td>
<td>16#600</td>
<td>crc-delimiter before talker and formatter received</td>
</tr>
<tr>
<td>FIELD_DELIM_TO_EARLY</td>
<td>16#700</td>
<td>field-delimiter before talker and formatter received</td>
</tr>
<tr>
<td>HEX_DELIM_TO_EARLY</td>
<td>16#800</td>
<td>code-delimiter for HEX representation before address-field received</td>
</tr>
<tr>
<td>TO_MANY_PARAMETERS</td>
<td>16#900</td>
<td>Constant NMEA_MAX_VALUES_PER_SENTENCE to small</td>
</tr>
<tr>
<td>PARAMETER_TO_LONG</td>
<td>16#A00</td>
<td>Constant NMEA_MAX_VALUE_LENGTH to small</td>
</tr>
<tr>
<td>INVALID_SENTENCE_TYPE</td>
<td>16#1000</td>
<td>invalid sentence type</td>
</tr>
<tr>
<td>INVALID_ADDRESSFIELD</td>
<td>16#2000</td>
<td>invalid address field -&gt; must have normally 5 char -&gt; for proprietary sentences only 4 char allowed</td>
</tr>
<tr>
<td>PIPE_INVALID_PARAMETER</td>
<td>16#8001</td>
<td></td>
</tr>
<tr>
<td>PIPE_NO_DATA</td>
<td>16#8002</td>
<td></td>
</tr>
<tr>
<td>PIPE_FULL</td>
<td>16#8003</td>
<td></td>
</tr>
</tbody>
</table>
### Parameter (PARAMS)

<table>
<thead>
<tr>
<th>Scope</th>
<th>Name</th>
<th>Type</th>
<th>Initial</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>NMEA_MAX_VALUES_PER_SENTENCE</td>
<td>UINT</td>
<td>20</td>
<td>this constant allow f.e. 20 parameter / values for each sentence</td>
</tr>
<tr>
<td></td>
<td>NMEA_MAX_VALUE_LENGTH</td>
<td>INT</td>
<td>80</td>
<td>this constant allow f.e. 80 ASCII-Characters for each parameter / value</td>
</tr>
<tr>
<td></td>
<td>NMEA_PIPE_SIZE</td>
<td>UINT</td>
<td>1024</td>
<td>size of the communication buffer for sending NMEA sentences</td>
</tr>
</tbody>
</table>

Note: It has to be big enough for one worst case sentence.
## VersionHistory (GVL)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>WagoAppSerial_NMEA</td>
<td>ProjectInfo</td>
</tr>
</tbody>
</table>

WagoAppSerial_NMEA
This is a dictionary of all referenced libraries and their name spaces.

**Standard**
*Library Identification:*
Placeholder: Standard
Default Resolution: Standard, * (System)
Namespace: Standard

*Library Properties:*

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

**SysMem**
*Library Identification:*
Placeholder: SysMem
Default Resolution: SysMem, * (System)
Namespace: SysMem

*Library Properties:*

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

**WagoAppCom**
*Library Identification:*
Placeholder: WagoAppCom
Default Resolution: WagoAppCom, * (WAGO)
Namespace: WagoAppCom

*Library Properties:*
• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

Library Parameter:
Parameter: CUIDEFAULTSYSTEMBUFFERSIZE = 1028

WagoSysErrorBase
Library Identification:
Placeholder: WagoSysErrorBase
Default Resolution: WagoSysErrorBase, * (WAGO)
Namespace: WagoSysErrorBase

Library Properties:

• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

Library Parameter:
Parameter: RES_LOG_MAX_FILESIZE = 2000
Parameter: RES_LOG_MAX_FILES = 1
Parameter: RES_LOG_MAX_ENTRIES = 200
Parameter: RES_LOG_NAME = 'WagoAppResultLogger'

WagoSysVersion
Library Identification:
Name: WagoSysVersion
Version: 1.0.0.0
Company: WAGO
Namespace: WagoSysVersion

Library Properties:

• LinkAllContent: False
• QualifiedOnly: False
• SystemLibrary: False
• Optional: False

WagoTypesErrorBase
Library Identification:
Placeholder: WagoTypesErrorBase
Default Resolution: WagoTypesErrorBase, * (WAGO)
Namespace: WagoTypesErrorBase
Library Properties:

- LinkAllContent: False
- QualifiedOnly: False
- SystemLibrary: False
- Optional: False

© WAGO Kontakttechnik GmbH & Co. KG, Germany 2018 – All rights reserved. For the avoidance of doubt, this copyright notice does not only apply to the information above but also and primarily to the described library itself. Please note that third-party products are always mentioned without reference to intellectual property rights, including patents, utility models, designs and trademarks, accordingly the existence of such rights cannot be excluded. WAGO is a registered trademark of WAGO Verwaltungsgesellschaft mbH.