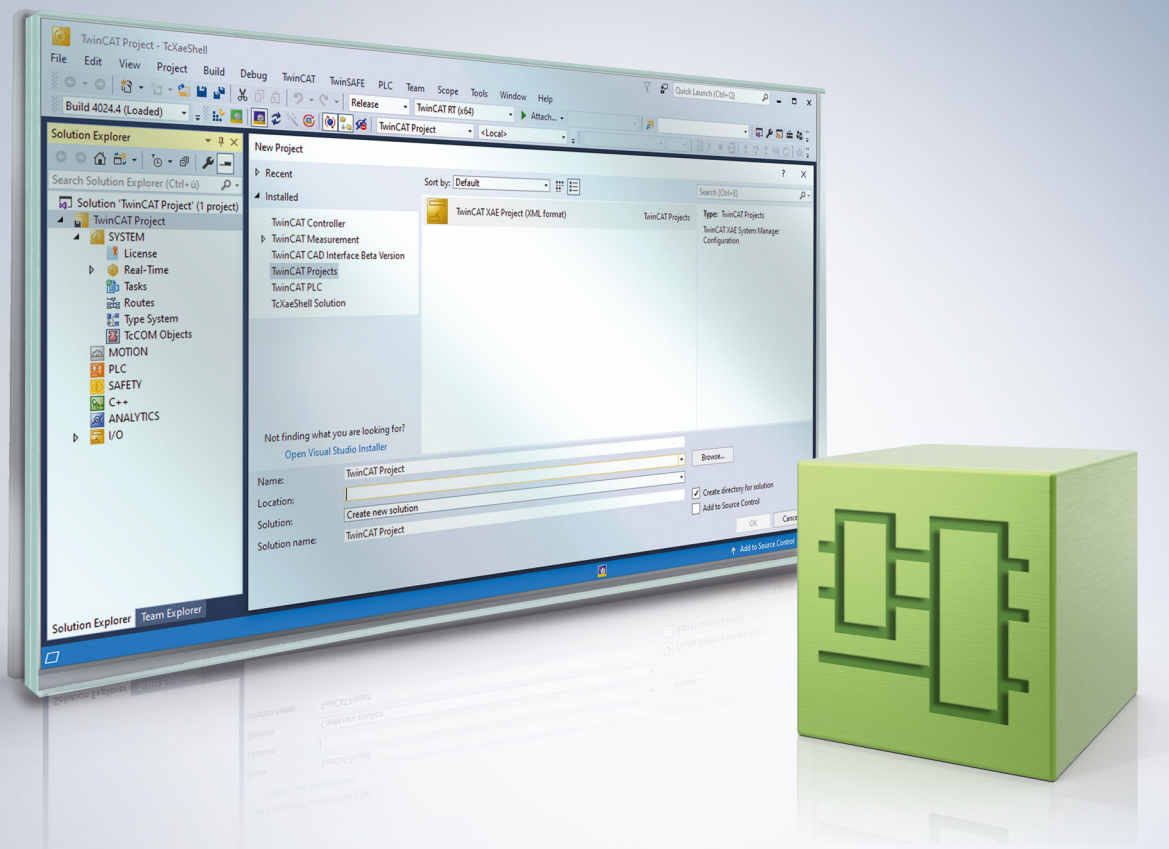
BECKHOFF New Automation Technology

Manual | EN

TE1000

TwinCAT 3 | PLC Library: Tc3_BA2_Common



23.10.2023 | Version: 1.0.1

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER

Hazard with high risk of death or serious injury.

⚠ WARNING

Hazard with medium risk of death or serious injury.

⚠ CAUTION

There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE

The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Introduction

The TwinCAT 3 Building Automation library (Tc3_BA2_Common) contains function blocks that are required for working with the TwinCAT Functions TF8020 as well as TF8040.

3 General Information

Further libraries required

For PC systems and Embedded PCs (CXxxxx):

- Tc2_IoFunctions
- Tc2_Standard
- Tc2_System
- Tc2_Utilities
- Tc2_DataExchange

4 Programming

4.1 POUs

4.1.1 Functions

4.1.1.1 Compare

4.1.1.1.1 F_BA_CompareVersion

The function F_BA_CompareVersion of return type BOOL compares two version numbers *stVersion1* and *stVersion2*, each of type ST_BA_Version [▶ 78]. The input variable *nLimit* specifies how many digits are to be compared, starting with 1 = "Major". The enumeration *eCompare* specifies the comparison operation. The function return value changes to TRUE if the comparison is fulfilled.

For example, if stVersion1 is smaller than stVersion2, then the comparison *eCompare* = *E_BA_CompareMode.eLower* returns a TRUE as function return.

Syntax

```
FUNCTION F_BA_CompareVersion : BOOL
VAR_INPUT
  stVersion1   : ST_BA_Version;
  stVersion2   : ST_BA_Version;

  eCompare     : E_BA_CompareMode  := E_BA_CompareMode.eEqual;
  nLimit       : UINT(1 .. 4)      := 4;
END_VAR
```

Inputs

Name	Type	Description
stVersion1, stVersion2	ST_BA_Version [▶ 78]	Version numbers to be compared.
eCompare	E_BA_CompareMode [▶ 50]	Comparison operation to be performed.
nLimit	UINT(1 ... 4)	Number of digits to be compared, starting with "Major".

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2 Memory

4.1.1.2.1 F_BA_ByteCmp

The function F_BA_ByteCmp of return type DINT compares the contents of a memory area in byte steps with a compare byte *nCompare*.

As soon as a byte is found within the memory area that is smaller in value than the comparison byte, the function aborts its comparison and assumes the return value "-1". If a byte is found that is larger than the comparison byte, the function also aborts and takes the return value "1". If, on the other hand, no difference is found, i.e. all bytes of the memory area to be examined are identical to the comparison byte, the function assumes the value "0" when the comparison is completed.

The input variable *pValue* marks the beginning of the memory area, the variable *nSize* the length.

In case of an incorrect input, i.e. *pValue* = 0 or *nSize* = 0, the function is also aborted immediately and takes "-1" as return value.

Syntax

```
FUNCTION F_BA_ByteCmp : DINT
VAR_INPUT
  pValue    : PVOID;
  nSize     : UXINT;
  nCompare  : BYTE;
END_VAR
```

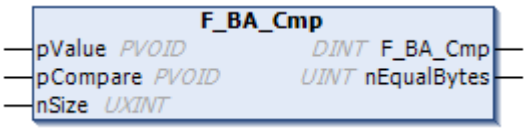
Inputs

Name	Type	Description
pValue	PVOID	Pointer to the beginning of the memory area to be examined.
nSize	UXINT	Length of the memory area.
nCompare	BYTE	Comparison byte.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.2 F_BA_Cmp



The function F_BA_Cmp of return type DINT compares two memory areas of the same size. Here *pValue* defines the beginning of the observation area and *pCompare* the beginning of the area to be compared. It is compared byte by byte. As soon as a byte is found in the observation area which is smaller than the one of the area to be compared, the function aborts the comparison and takes the return value "-1". If a byte is found in the observation area that is larger than that of the area to be compared, the function also aborts and takes the return value "1". If, on the other hand, no difference is found, i.e. all bytes of the observation area are identical to those of the area to be compared, the function assumes the value "0" at the end of the comparison.

The output *nEqualBytes* shows how many bytes were equal before the comparison operation was completed or aborted.

The input variable *nSize* defines the size of the two memory areas.

In case of an incorrect input, i.e. *pValue* = 0 or *nSize* = 0, the function is also aborted immediately and takes "-1" as return value.

Syntax

```
FUNCTION F_BA_Cmp : DINT
VAR_INPUT
  pValue          : PVOID;
  pCompare        : PVOID;
```

```
  nSize          : UXINT;
END_VAR
VAR_OUTPUT
  nEqualBytes    : UINT;
END_VAR
```

Inputs

Name	Type	Description
pValue	PVOID	Pointer to the beginning of the memory area to be examined.
pCompare	PVOID	Pointer to the beginning of the memory area to be compared.
nSize	UXINT	Length of the memory area.

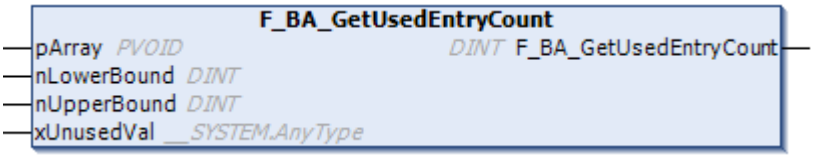
Outputs

Name	Type	Description
nEqualBytes	UINT	Display how many bytes were equal before the comparison operation was completed or aborted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.3 F_BA_GetUsedEntryCount



The function F_BA_GetUsedEntryCount of return type DINT searches the memory area of an ARRAY for the first entry marked as "unused" with xUnusedVal.

The memory area of the array is examined step by step starting from the address pArray: the step size is given by the size of xUnusedVal, the number of comparison steps by the difference of nLowerBound and nUpperBound. It is assumed that the entries nLowerBound and nUpperBound are correct.

The return value of the function shows how many elements do not match the value xUnusedVal until it is found. If the value is found, the search is aborted.

In case of an incorrect input, i.e. pArray = 0 or if the size of xUnusedVal = 0, the function is aborted immediately and takes "0" as return value.

Syntax

```
FUNCTION F_BA_GetUsedEntryCount : DINT
VAR_INPUT
  pArray         : PVOID;
  nLowerBound    : DINT;
  nUpperBound    : DINT;
  xUnusedVal     : ANY;
END_VAR
```

Inputs

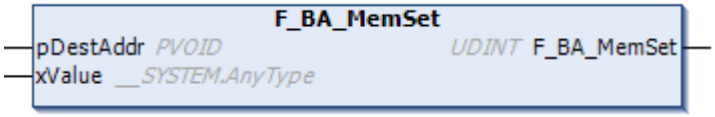
Name	Type	Description
pArray	PVOID	Pointer to the beginning of the memory area where the array to be examined is located.
nLowerBound	DINT	Lower range limit of the array.

Name	Type	Description
nUpperBound	DINT	Upper range limit of the array.
xUnusedVal	ANY	If an element of the array has the value specified here, it is considered "unused".

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.4 F_BA_MemSet



The function F_BA_MemSet of return type UDINT describes the address range starting with *pDestAddr* with the value of a variable *xValue* of any type.

The function return value itself is assigned the result of the internal MEMCPY function, i.e. "0" for incorrect copying and values greater than "0" for the number of bytes copied.

Syntax

```
FUNCTION F_BA_MemSet : UDINT
VAR_INPUT
  pDestAddr    : PVOID;
  xValue       : ANY;
END_VAR
```

Inputs

Name	Type	Description
pDestAddr	PVOID	Destination address of the write operation.
xValue	ANY	Value to be written.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.5 F_BA_MemSetEx

The function F_BA_MetSetEx of return type UDINT fills the address range starting with *pDestAddr* and length *nDestSize* with the value of a variable *xValue* of any type.

The prerequisite for this is that the defined filling area is larger by an integer multiple than the filling variable *xValue* itself: *nDestSize* = n * *xValue.diSize*.

The function return value itself is assigned the result of the internal MEMCPY function, i.e. "0" for incorrect copying and values greater than "0" for the number of bytes copied.

The value "0" is also returned if the defined filling area is not larger than the filling variable *xValue* by an integer multiple, and filling has not taken place.

Syntax

```
FUNCTION F_BA_MemSetEx : UDINT
VAR_INPUT
  pDestAddr    : PVOID;
  nDestSize    : UDINT;
  xValue       : ANY;
END_VAR
```

Inputs

Name	Type	Description
pDestAddr	PVOID	Destination address of the filling area.
nDestSize	UDINT	Size of the filling area.
xValue	ANY	Filling value

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.6 F_BA_OffsetPtr

The function F_BA_OffestPtr of return type PVOID adds an offset *nOffset* to the entered address *pAddr* and returns the result as return value of the function. This value then in turn represents an address.

The function distinguishes internally whether the runtime system used is of type x64 or x86: for the x64 system the offset is converted to a variable of type ULINT and added, for an x86 system to a variable of type UDINT.

Syntax

```
FUNCTION F_BA_OffsetPtr : PVOID
VAR_INPUT
  pAddr      : PVOID;
  nOffset    : DINT;
END_VAR
```

Inputs

Name	Type	Description
pAddr	PVOID	Base address
nOffset	DINT	Offset

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.2.7 F_BA_DiffPtr

The function F_BA_DiffPrt of return type UDINT specifies the difference between two memory addresses as an absolute value: internally the smaller address is subtracted from the larger one.

Syntax

```
FUNCTION F_BA_DiffPtr : UDINT
VAR_INPUT
  pAddr1    : PVOID;
  pAddr2    : PVOID;
END_VAR
```

Inputs

Name	Type	Description
pAddr1	PVOID	Address 1
pAddr2	PVOID	Address 2

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3 Types

4.1.1.3.1 AnyLEN

The function AnyLEN with the return type INT determines the described length of a string variable. This is applied to input STR of type ANY and can thus be analyzed.

For string variables, fixed memory areas are reserved depending on the declaration, but these are usually only partially filled with a text.

If the created variable is of type STRING, the memory area in which the variable is located is searched byte by byte until a byte with the value "0" is found, i.e. the end of the filling text is reached. The return value of the function is equal to the number of bytes searched, excluding the "0" found.

If no byte with the value "0" is found, a text fills the entire area of the string, the return value of the function then corresponds to the memory size of the created string in bytes.

If the created variable is not of type STRING, the memory size of the variable in bytes is assigned to the return value of the function.

Syntax

```
FUNCTION AnyLEN : INT
VAR_INPUT
  STR    : ANY;
END_VAR
```

Inputs

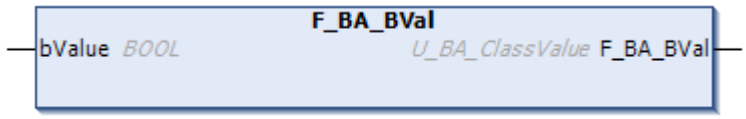
Name	Type	Description
STR	ANY	Variable to be examined. In principle, any simple variable type can be created here.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.2 ClassValue

4.1.1.3.2.1 F_BA_BVal



The return value of the F_BA_BVal function is created by the structure of type U_BA_ClassValue [▶ 80]. The input value *bValue* is written to the variable *bVal* of this structure. Due to the data type UNION, whose elements all start from the same memory address, all other values of this structure change as well.

Syntax

```
FUNCTION F_BA_BVal : U_BA_ClassValue
VAR_INPUT
  bValue    : BOOL;
END_VAR
```

Inputs

Name	Type	Description
bValue	BOOL	This value is assigned to *iVal* in the output structure.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.2.2 F_BA_ByteVal

The return value of the F_BA_ByteVal function is created by the structure of type U_BA_ClassValue [▶ 80]. The input values *nByte1*, *nByte2*, *nByte3* and *nByte4* are assigned in this structure on the respective values *nByteVal[1]*, *nByteVal[2]*, *nByteVal[3]* and *nByteVal[4]*. Due to the data type UNION, whose elements all start from the same memory address, all other values of this structure change as well.

Syntax

```
FUNCTION F_BA_ByteVal : U_BA_ClassValue
VAR_INPUT
  nByte1    : BYTE;
  nByte2    : BYTE;
  nByte3    : BYTE;
  nByte4    : BYTE;
END_VAR
```

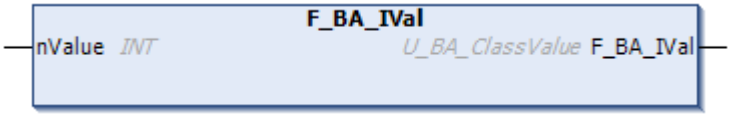
Inputs

Name	Type	Description
nByte1	BYTE	This value is assigned _nByteVal[1]_to the output structure.
nByte2	BYTE	This value is assigned _nByteVal[2]_to the output structure.
nByte3	BYTE	This value is assigned _nByteVal[3]_to the output structure.
nByte4	BYTE	This value is assigned _nByteVal[4]_to the output structure.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.2.3 F_BA_IVal



The return value of the F_BA_IVal function is created by the structure of type U_BA_ClassValue [▶ 80]. The input value *nValue* is written to the variable *iVal* of this structure. Due to the data type UNION, whose elements all start from the same memory address, all other values of this structure change as well.

Syntax

```
FUNCTION F_BA_IVal : U_BA_ClassValue
VAR_INPUT
  nValue    : INT;
END_VAR
```

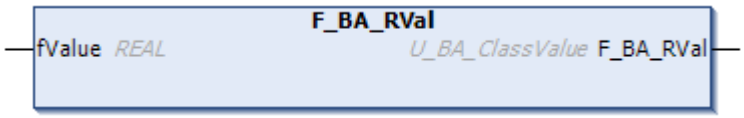
Inputs

Name	Type	Description
nValue	INT	This value is assigned to *iVal* in the output structure.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.2.4 F_BA_RVal



The return value of the F_BA_RVal function is created by the structure of type U_BA_ClassValue [▶ 80]. The input value *fValue* is written to the variable *rVal* of this structure. Due to the data type UNION, whose elements all start from the same memory address, all other values of this structure change as well.

Syntax

```
FUNCTION F_BA_RVal : U_BA_ClassValue
VAR_INPUT
  fValue    : REAL;
END_VAR
```

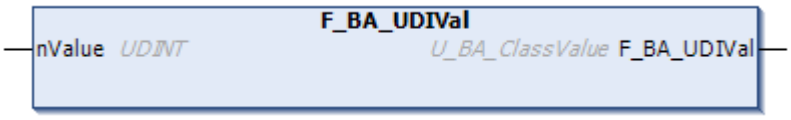
Inputs

Name	Type	Description
fValue	REAL	This value is assigned to *rVal* in the output structure.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.2.5 F_BA_UDIVal



The return value of the F_BA_UDIVal function is created by the structure of type U_BA_ClassValue [▶ 80]. The input value *nValue* is written to the variable *udiVal* of this structure. Due to the data type UNION, whose elements all start from the same memory address, all other values of this structure change as well.

Syntax

```
FUNCTION F_BA_UDIVal : U_BA_ClassValue
VAR_INPUT
  nValue    : BOOL;
END_VAR
```

Inputs

Name	Type	Description
nValue	UINT	This value is assigned to *udiVal* in the output structure.

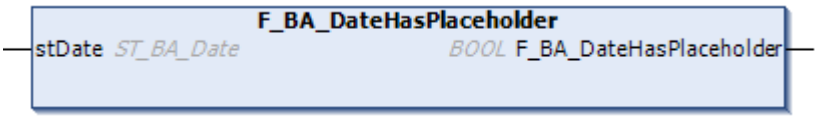
Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3 Date and Time

4.1.1.3.3.1 Check

4.1.1.3.3.1.1 F_BA_DateHasPlaceholder



The function F_BA_DateHasPlaceholder of return type BOOL returns a TRUE if from the input structure the year, month, day or weekday component holds the placeholder value 16#FF.

Syntax

```
FUNCTION F_BA_DateHasPlaceholder : BOOL
VAR_INPUT
  stDate    : ST_BA_Date;
END_VAR
```

Inputs

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Observed date-time entry.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.1.2 F_BA_DateUnspecified

The function F_BA_DataUnspecified of return type BOOL returns a TRUE if from the input structure the year, month, day and weekday components hold the placeholder value 16#FF.

Syntax

```
FUNCTION F_BA_DateUnspecified : BOOL
VAR_INPUT
  stDate    : ST_BA_Date;
END_VAR
```

Inputs

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Observed date-time entry.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.1.3 F_BA_IsLeapYear

The function F_BA_IsLeapYear of return type BOOL returns a TRUE if the entered year is a leap year.

Syntax

```
FUNCTION F_BA_IsLeapYear : BOOL
VAR_INPUT
  nYear  : UDINT;
END_VAR
```

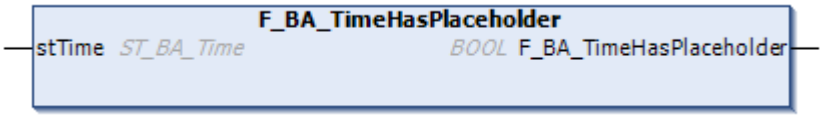
Inputs

Name	Type	Description
nYear	UDINT	Year to be studied.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.1.4 F_BA_TimeHasPlaceholder



The function F_BA_TimeHasPlaceholder of return type BOOL returns a TRUE if from the input structure the hour, minute or second component holds the placeholder value 16#FF.

Syntax

```
FUNCTION F_BA_TimeHasPlaceholder : BOOL
VAR_INPUT
  stTime    : ST_BA_Time;
END_VAR
```

Inputs

Name	Type	Description
stTime	ST_BA_Time [▶ 75]	Observed day-time entry.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.1.5 F_BA_TimeUnspecified

The function F_BA_TimeUnspecified of return type BOOL returns a TRUE if from the input structure the hour, minute and second components hold the placeholder value 16#FF.

Syntax

```
FUNCTION F_BA_TimeUnspecified : BOOL
VAR_INPUT
  stTime    : ST_BA_Time;
END_VAR
```

Inputs

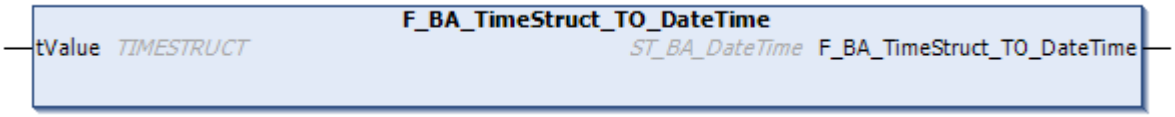
Name	Type	Description
stTime	ST_BA_Time [▶ 75]	Observed day-time entry.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2 Convert

4.1.1.3.3.2.1 F_BA_TimeStruct_TO_DateTime



The function F_BA_TimeStruct_TO_DateTime of return type ST_BA_DateTime [▶ 74] converts a time value *tValue* of type TIMESTRUCT into a time value of type ST_BA_DateTime [▶ 74].

Syntax

```
FUNCTION F_BA_TIMESTRUCT_TO_DateTime : ST_BA_DateTime
VAR_INPUT
  tValue    : TIMESTRUCT;
END_VAR
```

Inputs

Name	Type	Description
tValue	TIMESTRUCT	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.2 F_BA_TimeStruct_TO_Time

The function F_BA_TimeStruct_TO_Time of return type ST_BA_Time [▶ 75] converts a time value *tValue* of type TIMESTRUCT into a time value of type ST_BA_Time [▶ 75].

Syntax

```
FUNCTION F_BA_TIMESTRUCT_TO_Time : ST_BA_Time
VAR_INPUT
  tValue    : TIMESTRUCT;
END_VAR
```

Inputs

Name	Type	Description
tValue	TIMESTRUCT	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.3 F_BA_To100msDate

The function F_BA_To100msDate of return type UDINT converts a date *stBACnetDate* of type ST_BA_Date [▶ 74] into a tenths of a second value related to 01.01.1900 00:00:00.

Syntax

```
FUNCTION F_BA_To100msDate : UDINT
VAR_INPUT
  stBACnetDate    : ST_BA_Date;
END_VAR
```

Inputs

Name	Type	Description
stBACnetDate	ST_BA_Date [▶ 74]	Date input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.4 F_BA_To100msTime

The function F_BA_To100msTime of return type UDINT converts a time of day *stBACnetTime* of type ST_BA_Time [▶ 75] into tenths of seconds starting from 00:00:00.

Entries of the time input value *stBACnetTime*, which are unspecified, i.e. entries with the value 16#FF, are evaluated in the calculation with 0.

Syntax

```
FUNCTION F_BA_To100msTime : UDINT
VAR_INPUT
  stBACnetTime    : ST_BA_Time;
END_VAR
```

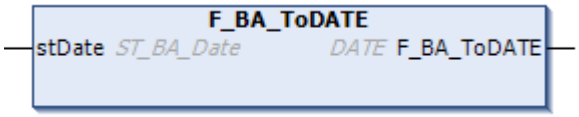
Inputs

Name	Type	Description
stBACnetTime	ST_BA_Time [▶ 75]	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.5 F_BA_ToDate



The function F_BA_ToDATE of return type DATE converts a date *stDate* of type ST_BA_Date [▶ 74] into a date of type DATE.

Syntax

```
FUNCTION F_BA_ToDATE : DATE
VAR_INPUT
  stDate    : ST_BA_Date;
END_VAR
```

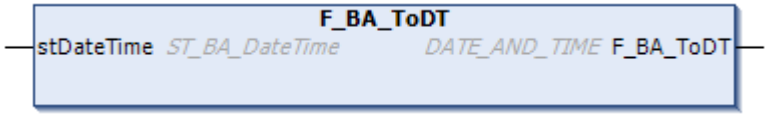
Inputs

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Date input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.6 F_BA_ToDT



The function F_BA_ToDT of return type DATE converts a date *stDate* of type ST_BA_Date [▶ 74] into a date of type DATE.

Syntax

```
FUNCTION F_BA_ToDATE : DATE
VAR_INPUT
  stDate    : ST_BA_Date;
END_VAR
```

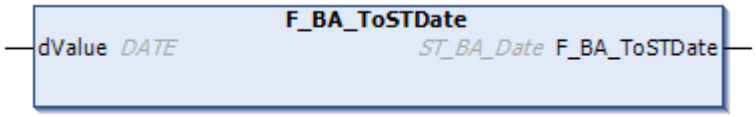
Inputs

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Date input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.7 F_BA_ToSTDate



The function F_BA_ToSTDate of return type ST_BA_Date [▶ 74] converts a date *dValue* of type DATE into a date of type ST_BA_Date [▶ 74].

Syntax

```
FUNCTION F_BA_ToSTDate : ST_BA_Date
VAR_INPUT
  dValue    : DATE;
END_VAR
```

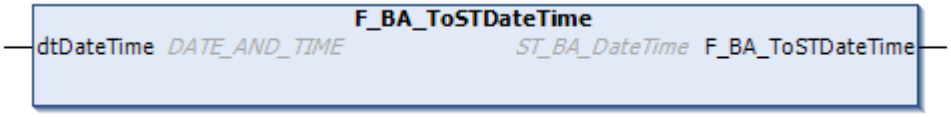
Inputs

Name	Type	Description
dValue	DATE	Date input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.8 F_BA_ToSTDateTime



The function F_BA_ToSTDateTime of return type ST_BA_DateTime [▶ 74] converts a time value *dtDateTime* of type DT (DATE_AND_TIME) into a time value of type ST_BA_DateTime [▶ 74].

Syntax

```
FUNCTION  F_BA_ToSTDateTime  :  ST_BA_DateTime
VAR_INPUT
    dtDateTime    :  DT;
END_VAR
```

Inputs

Name	Type	Description
dtDateTime	DT	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.9 F_BA_ToSTTime

The function F_BA_ToSTTime of return type ST_BA_Time [▶ 75] converts a time value *tTime* of type TIME into a time value of type ST_BA_Time [▶ 75].

Syntax

```
FUNCTION F_BA_ToSTTime : ST_BA_Time
VAR_INPUT
  tTime    : TIME;
END_VAR
```

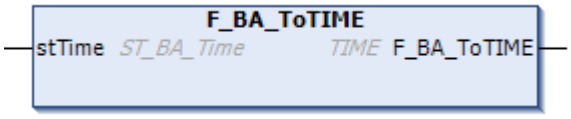
Inputs

Name	Type	Description
tTime	TIME	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.2.10 F_BA_ToTime



The function F_BA_ToTime of return type TIME converts a time value *stTime* of type ST_BA_Time into a time value of type TIME.

Entries of the time input value *stTime*, which are unspecified, i.e. entries with the value 16#FF, are evaluated in the calculation with 0.

Syntax

```
FUNCTION F_BA_ToTIME : TIME
VAR_INPUT
  stTime     : ST_BA_Time;
END_VAR
```

Inputs

Name	Type	Description
stTime	ST_BA_Time [▶ 75]	Time input value to be converted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.3 F_BA_CountLeapYears

The function F_BA_CountLeapYears of return type UDINT calculates the number of leap years from the entered start year *nStartYear* to the end of the period under consideration *nYear* (inclusive).

Syntax

```
FUNCTION F_BA_CountLeapYears : UDINT
VAR_INPUT
  nYear         : UDINT;
  nStartYear    : UDINT    := 0;
END_VAR
```

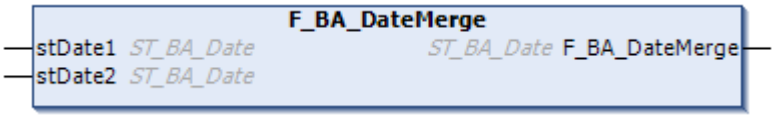
Inputs

Name	Type	Description
nYear	UDINT	End of the period under consideration.
nStartYear	UDINT	Start of the period under consideration.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.4 F_BA_DateMerge



The function F_BA_DateMerge of the return type ST_BA_Date [▶ 74] replaces those subcomponents of input date *stDate1* which are evaluated as not specified (16#FF) with the corresponding components of input date *stDate2*.

Syntax

```
FUNCTION F_BA_DateMerge : ST_BA_Date
VAR_INPUT
  stDate1       : ST_BA_Date;
  stDate2       : ST_BA_Date;
END_VAR
```

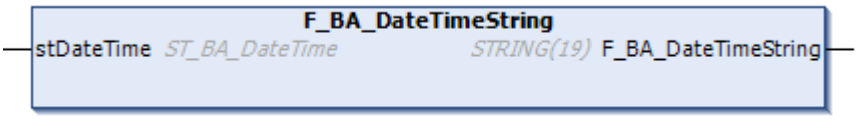
Inputs

Name	Type	Description
stDate1	ST_BA_Date [▶ 74]	The input date, which is checked if parts need to be replaced.
stDate2	ST_BA_Date [▶ 74]	Input date, which is used to replace *stDate1*.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.5 F_BA_DateTimeString



The function F_BA_DateTimeString of return type STRING(19) converts a date *stDateTime* into a STRING of format DD.MM.YYYY hh:mm:ss.

Decimal values smaller than 10 are preceded by a "0", e.g. the month September is displayed with "09" instead of simply "9".

Implausible values (e.g. month > 12) or entries evaluated as unspecified, i.e. with 16#FF, are output with "**" or "****" for the year specification.

Syntax

```
FUNCTION F_BA_DateTimeString : STRING(19)
VAR_IN_OUT
  stDateTime    : ST_BA_DateTime;
END_VAR
```

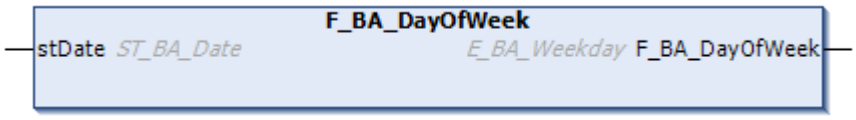
Inputs/outputs

Name	Type	Description
stDateTime	ST_BA_DateTime [▶ 74]	Date and time from which the return string is formed.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.6 F_BA_DayOfWeek



The function F_BA_DayOfWeek of return type E_BA_Weekday [▶ 56] determines the corresponding weekday from a given date *stDate*.

Syntax

```
FUNCTION F_BA_DayOfWeek : E_BA_Weekday
VAR_INPUT
  stDate    : ST_BA_Date;
END_VAR
```

Inputs

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Date from which the day of the week is determined.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.7 F_BA_DaysInMonth

The function F_BA_DaysInMonth of return type UDINT determines from a given date *stBACnetDate* the number of days of the given month, provided that the month and the year are not evaluated as unspecified, i.e. with 16#FF. In case of missing specification the return value of the function is 16#FFFFFF.

Syntax

```
FUNCTION F_BA_DaysInMonth : UDINT
VAR_INPUT
  stBACnetDate    : ST_BA_Date;
END_VAR
```

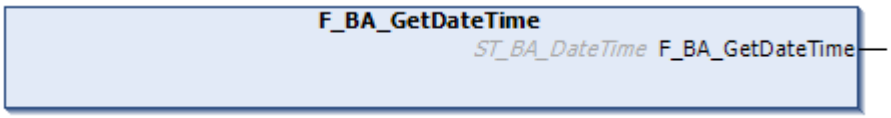
Inputs

Name	Type	Description
stBACnetDate	ST_BA_Date [▶ 74]	Date from whose month the number of days is to be determined.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.8 F_BA_GetDateTime



The return type ST_BA_DateTime [▶ 74] function outputs the date and time.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

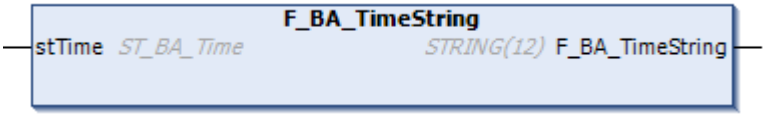
4.1.1.3.3.9 F_BA_GetDT

The return type DT (DATE_AND_TIME) function outputs the date and time.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.10 F_BA_TimeMerge



The function F_BA_TimeString of return type ST_BA_Time [▶ 75] replaces those subcomponents of the input time stamp *stTime1* which are evaluated as not specified (16#FF) with the corresponding components of the input time stamp *stTime2*.

Syntax

```
FUNCTION F_BA_TimeMerge : ST_BA_Time
VAR_INPUT
  stTime1    : ST_BA_Time;
  stTime2    : ST_BA_Time;
END_VAR
```

Inputs

Name	Type	Description
stTime1	ST_BA_Time [▶ 75]	Input timestamp, which is checked.
stTime2	ST_BA_Time [▶ 75]	Input timestamp which is used to replace unspecified parts of *stTime1*.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.3.11 F_BA_TimeString

The function F_BA_TimeString of return type STRING(12) converts a timestamp *stTime* into a STRING in the format hh:mm:ss:**.
The last two digits of the hundredths of a second are ignored and filled with "*".
Decimal values smaller than 10 are preceded by a "0", e.g. the ninth hour in the morning is represented with "09" instead of simply "9".

Implausible values (e.g. second > 59) or entries evaluated as unspecified, i.e. with 16#FF, are output with "**".

Syntax

```
FUNCTION F_BA_TimeString : STRING(12)
VAR_IN_OUT
  stTime    : ST_BA_Time;
END_VAR
```

Inputs/outputs

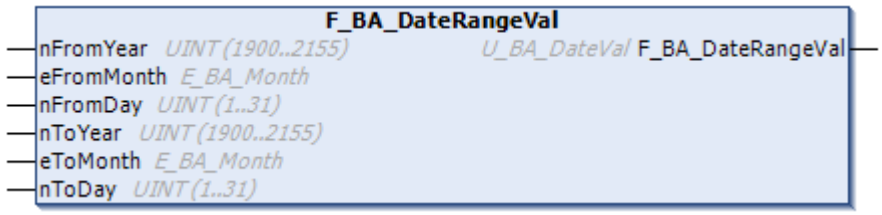
Name	Type	Description
stTime	ST_BA_Time [▶ 75]	Timestamp from which the return string is formed.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.4 DateValue

4.1.1.3.4.1 F_BA_DateRangeVal



The function F_BA_DateRangeVal of return type U_BA_DateVal [▶ 76] uses the input variables *nFromYear*, *eFromMonth*, *nFromDay*, *nToYear*, *eToMonth* and *nToDay* fills the subcomponent ST_BA_DateRange [▶ 74] of the function return U_BA_DateVal [▶ 76]. This defines a time range.

Syntax

```
FUNCTION F_BA_DateRangeVal : U_BA_DateVal
VAR_INPUT
  nFromYear     : UINT(1900 .. 2155);
  eFromMonth    : E_BA_Month := E_BA_Month.Unspecified;
  nFromDay      : UINT(1 .. 31);

  nToYear       : UINT(1900 .. 2155);
  eToMonth      : E_BA_Month := E_BA_Month.Unspecified;
  nToDay        : UINT(1 .. 31);
END_VAR
```

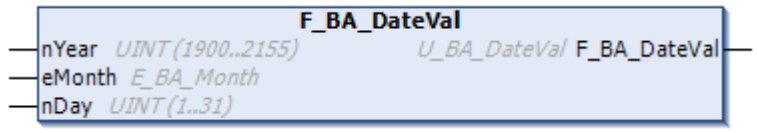
Inputs

Name	Type	Description
nFromYear	UINT	Starting year of the time range.
eFromMonth	E_BA_Month [▶ 55]	Starting month of the time range.
nFromDay	UINT	Start day of the time range.
nToYear	UINT	End year of the time range.
eToMonth	E_BA_Month [▶ 55]	End month of the time range.
nToDay	UINT	End tag of the time range.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.4.2 F_BA_DateVal



The function F_BA_DateVal of the return type U_BA_DateVal [▶ 76] fills the subcomponent ST_BA_Date [▶ 74] of the function return U_BA_DateVal [▶ 76] using the input variables *nYear*, *eMonth* and *nDay*. This defines a date.

Syntax

```
FUNCTION F_BA_DateVal : U_BA_DateVal
VAR_INPUT
  nYear     : UINT(1900 .. 2155);
```

```
  eMonth    : E_BA_Month := E_BA_Month.Unspecified;
  nDay      : UINT(1 .. 31);
END_VAR
```

Inputs

Name	Type	Description
nYear	UINT	Entry of the year.
eMonth	E_BA_Month [▶ 55]	Entry of the month.
nDay	UINT	Entry of the day.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.4.3 F_BA_WeekNDayVal

The function F_BA_WeekNDayVal of the return type U_BA_DateVal [▶ 76] fills the subcomponent ST_BA_WeekNDay [▶ 75] of the function return U_BA_DateVal [▶ 76] using the input variables *eWeekday*, *eWeekOfMonth* and *eMonth*. This defines a day of the week within a month.

Syntax

```
FUNCTION F_BA_WeekNDayVal : U_BA_DateVal
VAR_INPUT
  eWeekday        : E_BA_Weekday  := E_BA_Weekday.Invalid;
  eWeekOfMonth    : E_BA_Week     := E_BA_Week.Invalid;
  eMonth          : E_BA_Month    := E_BA_Month.Invalid;
END_VAR
```

Inputs

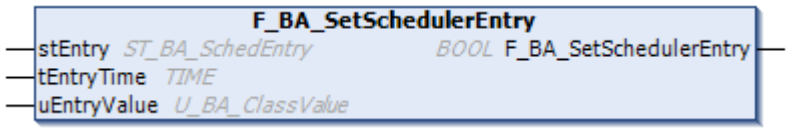
Name	Type	Description
eWeekday	E_BA_Weekday [▶ 56]	Entry of the day of the week.
eWeekOfMonth	E_BA_Week [▶ 55]	Entry of the week within the month.
eMonth	E_BA_Month [▶ 55]	Entry of the month.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.5 Scheduler

4.1.1.3.5.1 F_BA_SetSchedulerEntry



The function F_BA_SetSchedulerEntry of return type BOOL first checks whether the input structure *uEntryValue* is undefined, i.e. filled with 16#FF values.
If this is the case, the value E_BA_SchedEntryState.eNull is assigned to the status variable *eState* of the input-output structure *stEntry*, in the opposite case the value E_BA_SchedEntryState.eValue.

The time component *stTime* of the input-output structure *stEntry* is assigned the input *tEntryTime*, adjusted via the function F_BA_ToSTTime [▶ 22] and the value component *uValue* is assigned the input value *uEntryValue*.
Nothing is assigned to the function return value itself.

Syntax

```
FUNCTION F_BA_SetSchedulerEntry : BOOL
VAR_IN_OUT
  stEntry        : ST_BA_SchedEntry;
END_VAR
VAR_INPUT
  tEntryTime     : TIME;
  uEntryValue    : U_BA_ClassValue;
END_VAR
```

Inputs/outputs

Name	Type	Description
stEntry	ST_BA_SchedEntry [▶ 77]	Reference to the schedule entry structure to be read and written.

Inputs

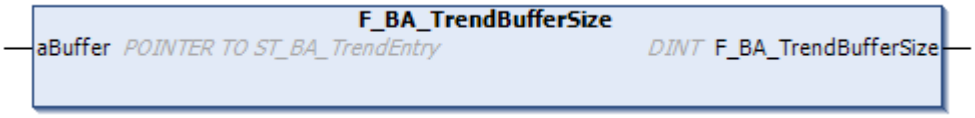
Name	Type	Description
tEntryTime	TIME	Input of the time.
uEntryValue	U_BA_ClassValue [▶ 80]	Input of the value.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.6 Trend

4.1.1.3.6.1 F_BA_TrendBufferSize



The function F_BA_TrendBufferSize of return type DINT determines the size of a trend, which is defined as ARRAY of type ST_BA_TrendEntry [▶ 79].

The input/output variable *aBuffer* refers to the trend to be examined and allows the linking of fields of undefined size by the syntax ARRAY [*].

Syntax

```
FUNCTION F_BA_TrendBufferSize : DINT
VAR_IN_OUT
  aBuffer    : ARRAY [*] OF ST_BA_TrendEntry;
END_VAR
```

Inputs/outputs

Name	Type	Description
aBuffer	ST_BA_TrendEntry [▶ 79]	Reference to the trend to be examined.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.3.7 F_BA_IsDisturbed

The function F_BA_IsDisturbed of return type BOOL checks the subcomponents of the input structure *stStateFlags*.

If the variable *bFault* or *bInAlarm* is set, the return value of the function is TRUE.

Syntax

```
FUNCTION F_BA_IsDisturbed : BOOL
VAR_INPUT
  stStateFlags    : ST_BA_StatusFlags;
END_VAR
```

Inputs

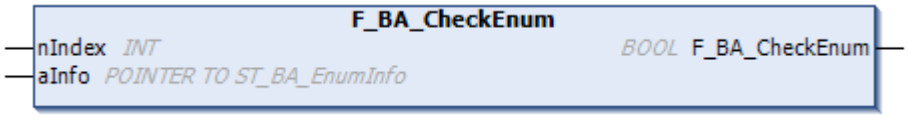
Name	Type	Description
stStateFlags	ST_BA_StatusFlags [▶ 78]	Status flags that are checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.4 Universal

4.1.1.4.1 F_BA_CheckEnum



The function F_BA_CheckEnum of return type BOOL determines whether the value *nIndex* of an enumeration exists within a predefined list *aInfo* of type ARRAY [*] OF ST_BA_EnumInfo [▶ 77].

By the syntax ARRAY [*] the reference to fields of undefined size is possible.

Sample:

```
aAction  : ARRAY[1 .. 2] OF ST_BA_EnumInfo := [
  (*eDirect*)  (sName := 'Direkt',    sDescription := 'Gleichläufiger Wirksinn',  sShortcut := ''),
  (*eReverse*) (sName := 'Indirekt',  sDescription := 'Gegenläufiger Wirksinn',   sShortcut := '')
  ];
```

If this field would be applied to the IN_OUT variable *aInfo*, the function would return "TRUE" for the values 1 and 2 at *nIndex*, otherwise "FALSE".

Syntax

```
FUNCTION F_BA_CheckEnum : BOOL
VAR_INPUT
  nIndex      : INT;
END_VAR
VAR_IN_OUT
  aInfo       : ARRAY [*] OF ST_BA_EnumInfo;
END_VAR
```

Inputs

Name	Type	Description
nIndex	INT	Index to be examined.

Inputs/outputs

Name	Type	Description
aInfo	ST_BA_EnumInfo	List in which the element with the index *nIndex* is to be found.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.4.2 TcLog

4.1.1.4.2.1 FB_BA_LogMessage

Internally the function FB_BA_LogMessage works with the function block FB_FormatString.
In the first run a message string is generated from *sLogText*. The text *sLogText* needs a valid format specification in addition to the message text.
The resulting text is preceded by a text argument in square brackets in the second run of FB_FormatString.

Sample

LogMessage

```
1   PROGRAM LogMessage
2   VAR
3       i               :   UINT;
4       sText1          :   T_MaxString := 'Message';
5   END_VAR
6
```

```
1   IF i < 10 THEN
2   F_BA_LogMessage(ADSLOG_MSGTYPE_ERROR,TO_STRING(i), sText1);
3   END_IF
4
5   i:=i+1;
6
7   IF i >= 10 THEN
8       i := 10;
9   END_IF
```

The F_BA_LogMessage function is run in ten consecutive cycles, then no more. The log type is ADSLOG_MSGTYPE_ERROR, which means that the message output is of the error message type. The variable *sLogCode* is assigned the control variable *i*, which appears in the text in square brackets. The input variable *sLogText* of the function contains the text "Message".

Output

Error List

Entire Solution | 10 Errors | 0 of 1 Warning | 0 Messages | Clear | Build + IntelliSense

Code	Description
	30.08.2023 11:33:22 969 ms | 'PlcTask' (350): [0] Message
	30.08.2023 11:33:23 014 ms | 'PlcTask' (350): [1] Message
	30.08.2023 11:33:23 059 ms | 'PlcTask' (350): [2] Message
	30.08.2023 11:33:23 104 ms | 'PlcTask' (350): [3] Message
	30.08.2023 11:33:23 149 ms | 'PlcTask' (350): [4] Message
	30.08.2023 11:33:23 194 ms | 'PlcTask' (350): [5] Message
	30.08.2023 11:33:23 239 ms | 'PlcTask' (350): [6] Message
	30.08.2023 11:33:23 284 ms | 'PlcTask' (350): [7] Message
	30.08.2023 11:33:23 329 ms | 'PlcTask' (350): [8] Message
	30.08.2023 11:33:23 374 ms | 'PlcTask' (350): [9] Message

The generated code is used to identify a specific location in the source code. The sample shown above serves to illustrate how this function works. Basically, a distinctive abbreviation (e.g. "IO50" if a message is issued within the method *InitObject* in line 50) should be explicitly specified for the generation of the log messages.

This will display an error at runtime and the shortcut "Ctrl+F" will reach the erroneous location.

Syntax

```
FUNCTION F_BA_LogMessage
VAR_INPUT
  nLogType    : DWORD     := ADSLOG_MSGTYPE_ERROR;
  sLogCode    : STRING    := '';
  sLogText    : T_MaxString;
END_VAR
```

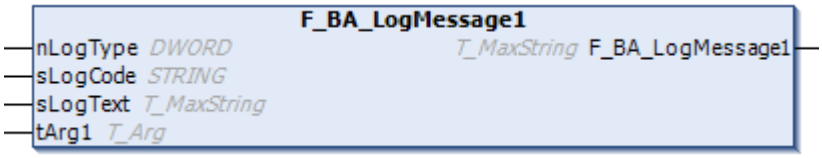
Inputs

Name	Type	Description
nLogType	DWORD	Log type, which can be set as a mask. The message is then output depending on the setting of this mask. The mask for ADSLOG_MSGTYPE_LOG is set internally as well.
sLogCode	STRING	An argument in textual form that precedes the message.
sLogText	T_MaxString	Message as text

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.4.2.2 FB_BA_LogMessage1



Internally the function FB_BA_LogMessage1 works with the function block FB_FormatString.
In the first run a message string is generated from *sLogText*, which is assigned to *tArg1*. The assigned variable must be linked with the correct data type conversion to *tArg1*. The text *sLogText* needs a valid format specification in addition to the message text.
The resulting text is preceded by a text argument in square brackets in the second run of FB_FormatString.

Sample

```
PROGRAM LogMessage1
VAR
    i               :   UINT;
    sText1          :   T_MaxString := 'Appendix';
END_VAR
```

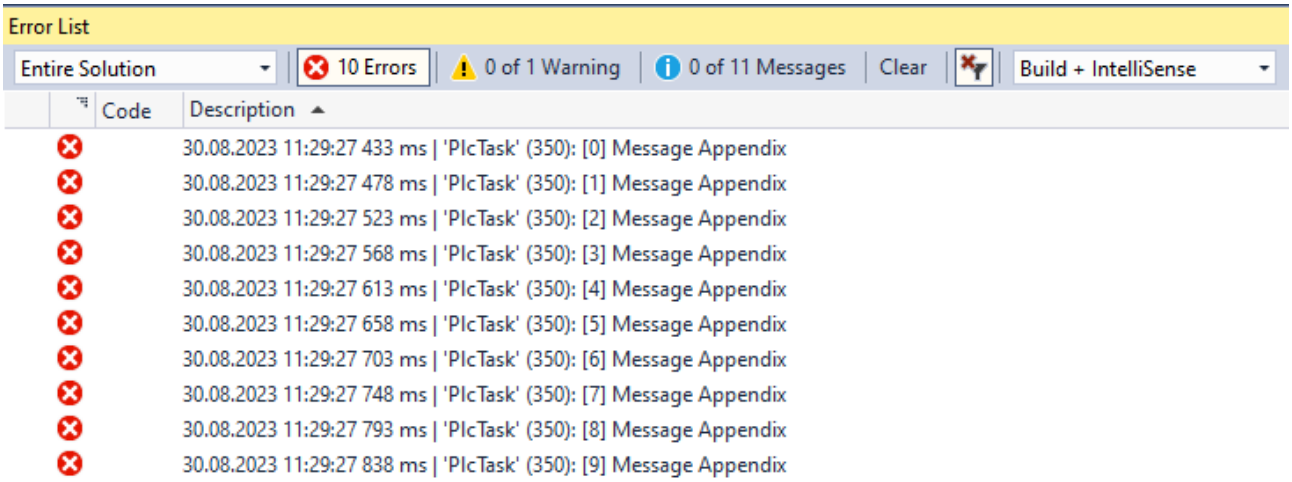
```
IF i < 10 THEN
    F_BA_LogMessage1(ADSLOG_MSGTYPE_ERROR,TO_STRING(i), 'Message %s', F_STRING(sText1));
END_IF

i:=i+1;

IF i >= 10 THEN
    i := 10;
END_IF
```

The F_BA_LogMessage1 function is run in ten consecutive cycles, then no more. The log type is ADSLOG_MSGTYPE_ERROR, which means that the message output is of the error message type. The variable *sLogCode* is assigned the control variable *i*, which appears in the text in square brackets. In addition to the text "Message" the input variable *sLogText* of the function also contains a format specification "%s", which also displays the text "Appendix" of the variable *sText1*. Without the format specification " Appendix " would not appear.

Output



The generated code is used to identify a specific location in the source code. The sample shown above serves to illustrate how this function works. Basically, a distinctive abbreviation (e.g. "IO50" if a message is issued within the method *InitObject* in line 50) should be explicitly specified for the generation of the log messages.

This will display an error at runtime and the shortcut "Ctrl+F" will reach the erroneous location.

Syntax

```
FUNCTION F_BA_LogMessage
VAR_INPUT
  nLogType    : DWORD     := ADSLOG_MSGTYPE_ERROR;
  sLogCode    : STRING    := '';
  sLogText    : T_MaxString;
  tArg1       : T_Arg;
END_VAR
```

Inputs

Name	Type	Description
nLogType	DWORD	Log type, which can be set as a mask. The message is then output depending on the setting of this mask. The mask for ADSLOG_MSGTYPE_LOG is set internally as well.
sLogCode	STRING	An argument in textual form that precedes the message.
sLogText	T_MaxString	Message as text.
tArg1	T_Arg	Further message text, which is placed at the end.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.4.2.3 FB_BA_LogMessage2

This function works like F_BA_LogMessage1 [▶ 34], but two message texts tArg1 and tArg2 can be added at the end.

Following the sample program from F_BA_LogMessage1 [▶ 34], a second "%s" must be added to the end of the message to display the additional message.

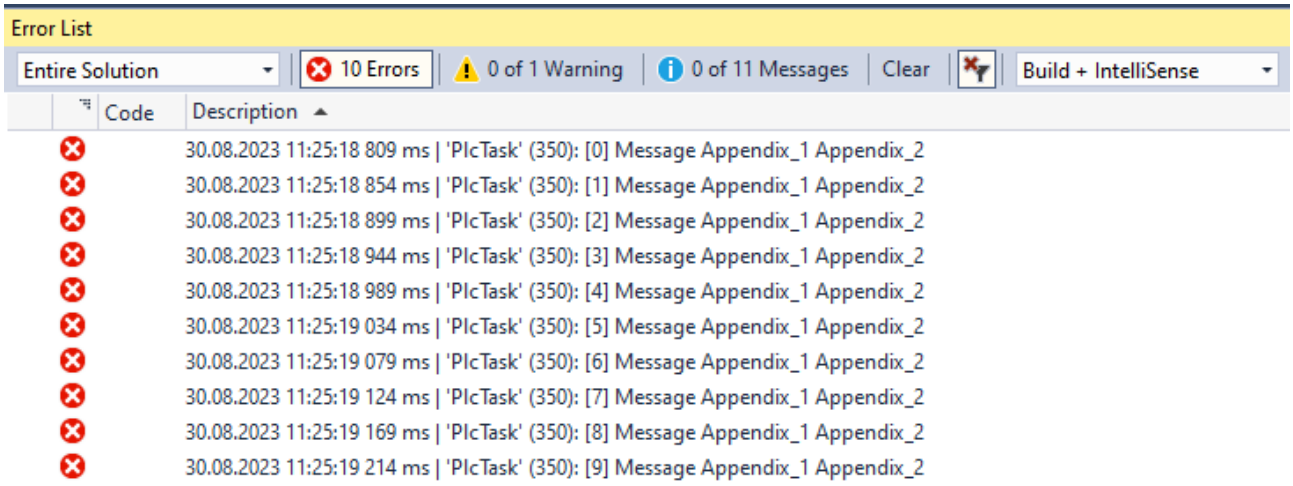
Sample

```
LogMessage2
1  PROGRAM LogMessage2
2  VAR
3      i               :   UINT;
4      sText1          :   T_MaxString := 'Appendix_1';
5      sText2          :   T_MaxString := 'Appendix_2';
6  END_VAR
7
```

```
1  IF i < 10 THEN
2      F_BA_LogMessage2(ADSLOG_MSGTYPE_ERROR,TO_STRING(i), 'Message %s %s', F_STRING(sText1), F_STRING(sText2));
3  END_IF
4
5  i:=i+1;
6
7  IF i >= 10 THEN
8      i := 10;
9  END_IF
```

There are eight more blocks of the same type: F_BA_LogMessage3 to F_BA_LogMessage10. With these functions it is possible to add three to ten additional messages. For each message, another text variable must then be created, *sText3* to *sText10* and a "%s" must be added to the message in line two of the above sample in each case.

Output



The generated code is used to identify a specific location in the source code. The sample shown above serves to illustrate how this function works. Basically, a distinctive abbreviation (e.g. "IO50" if a message is issued within the method *InitObject* in line 50) should be explicitly specified for the generation of the log messages.

This will display an error at runtime and the shortcut "Ctrl+F" will reach the erroneous location.

Syntax

```
FUNCTION F_BA_LogMessage
VAR_INPUT
  nLogType     : DWORD     := ADSLOG_MSGTYPE_ERROR;
  sLogCode     : STRING    := '';
  sLogText     : T_MaxString;
  tArg1        : T_Arg;
  tArg2        : T_Arg;
END_VAR
```

Inputs

Name	Type	Description
nLogType	DWORD	Log type, which can be set as a mask. The message is then output depending on the setting of this mask. The mask for ADSLOG_MSGTYPE_LOG is set internally as well.
sLogCode	STRING	An argument in textual form that precedes the message.
sLogText	T_MaxString	Message as text.
tArg1	T_Arg	Further message text, which is placed at the end.
tArg2	T_Arg	Further message text, which is placed at the end.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.4.2.4 FB_BA_LogMessage3

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.5 FB_BA_LogMessage4

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.6 FB_BA_LogMessage5

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.7 FB_BA_LogMessage6

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.8 FB_BA_LogMessage7

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.9 FB_BA_LogMessage8

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.10 FB_BA_LogMessage9

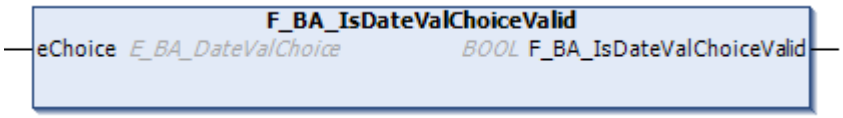
Application see F_BA_LogMessage2 [▶ 35].

4.1.1.4.2.11 FB_BA_LogMessage10

Application see F_BA_LogMessage2 [▶ 35].

4.1.1.5 Validation functions

4.1.1.5.1 F_BA_IsDateValChoiceValid



The F_BA_IsDateValChoiceValid function of return type BOOL checks whether the value of the enumeration applied to input *eChoice* is within the limits First and Last as defined at E_BA_DateValChoice [▶ 53].

Syntax

```
FUNCTION F_BA_IsDateValChoiceValid : BOOL
VAR_INPUT
  eChoice    : E_BA_DateValChoice;
END_VAR
```

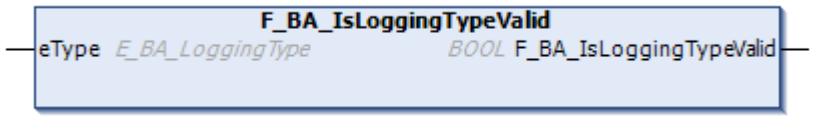
Inputs

Name	Type	Description
eChoice	E_BA_DateValChoice [▶ 53]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.2 F_BA_IsLoggingTypeValid



The F_BA_IsLoggingTypeValid function of return type BOOL checks whether the value of the enumeration applied to input *eType* is within the limits First and Last as defined under E_BA_LoggingType.

```
Syntax
FUNCTION F_BA_IsLoggingTypeValid : BOOL
VAR_INPUT
  eType    : E_BA_LoggingType;
END_VAR
```

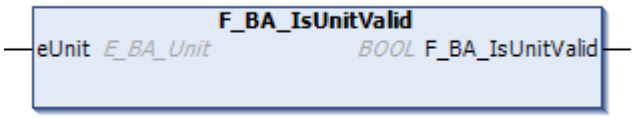
Inputs

Name	Type	Description
eType	E_BA_LoggingType [▶ 59]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.3 F_BA_IsUnitValid



The function F_BA_IsUnitValid of return type BOOL checks whether the value of the enumeration applied to the input *eUnit* is within the limits First and Last as defined at E_BA_Unit [▶ 60].

Syntax

```
FUNCTION F_BA_IsUnitValid : BOOL
VAR_INPUT
  eUnit    : E_BA_Unit;
END_VAR
```

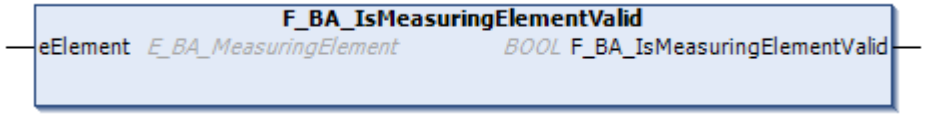
Inputs

Name	Type	Description
eUnit	E_BA_Unit [▶ 60]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.4 F_BA_IsMeasuringElementValid



The F_BA_IsMeasuringElementValid function of return type BOOL checks whether the value of the enumeration applied to input *eElement* is within the limits First and Last as defined at E_BA_MeasuringElement [▶ 70].

Syntax

```
FUNCTION F_BA_IsMeasuringElementValid : BOOL
VAR_INPUT
  eElement    : E_BA_MeasuringElement;
END_VAR
```

Inputs

Name	Type	Description
eElement	E_BA_MeasuringElement [▶ 70]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.5 F_BA_IsDataClassValid

The F_BA_IsDataClassValid function of return type BOOL checks whether the value of the enumeration applied to input *eClass* is within the limits First and Last as defined at E_BA_DataClass [▶ 56].

Syntax

```
FUNCTION F_BA_IsDataClassValid : BOOL
VAR_INPUT
  eClass    : E_BA_DataClass;
END_VAR
```

Inputs

Name	Type	Description
eClass	E_BA_DataClass [▶ 56]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.6 F_BA_IsDataTypeValid

The F_BA_IsDataTypeValid function of return type BOOL checks whether the value of the enumeration applied to input *eType* is within the limits First and Last as defined at E_BA_DataType [▶ 57].

Syntax

```
FUNCTION F_BA_IsDataTypeValid : BOOL
VAR_INPUT
  eType    : E_BA_DataType;
END_VAR
```

Inputs

Name	Type	Description
eType	E_BA_DataType [▶ 57]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.1.5.7 F_BA_IsWeekdayValid

The F_BA_IsWeekdayValid function of return type BOOL checks whether the value of the enumeration applied to input *eDay* is within the limits First and Last as defined at E_BA_Weekday [▶ 56].

Syntax

```
FUNCTION F_BA_IsWeekdayValid : BOOL
VAR_INPUT
  eDay  : E_BA_Weekday;
END_VAR
```

Inputs

Name	Type	Description
eDay	E_BA_Weekday [▶ 56]	Enumeration to be checked.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2 Function blocks

4.1.2.1 Controller

4.1.2.1.1 FB_BA_PIDCtrl

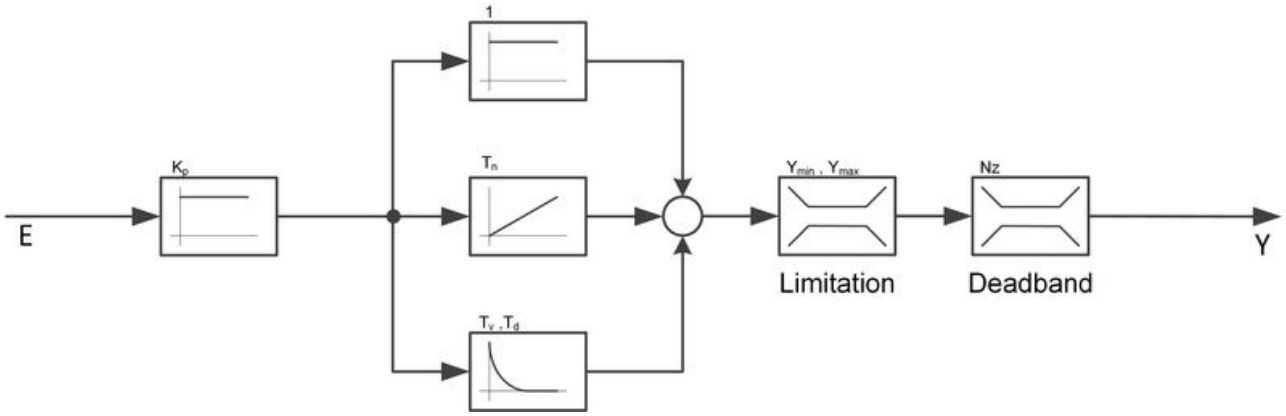
The FB_BA_PIDCtrl function block is a universal PID controller.

The controller is divided internally into two consecutive parts:

- the controller itself, illustrated in the functional diagrams below as P, I and D part with an output limitation.
- a deadband element (neutral zone) that applies a hysteresis to the output changes of the controller.

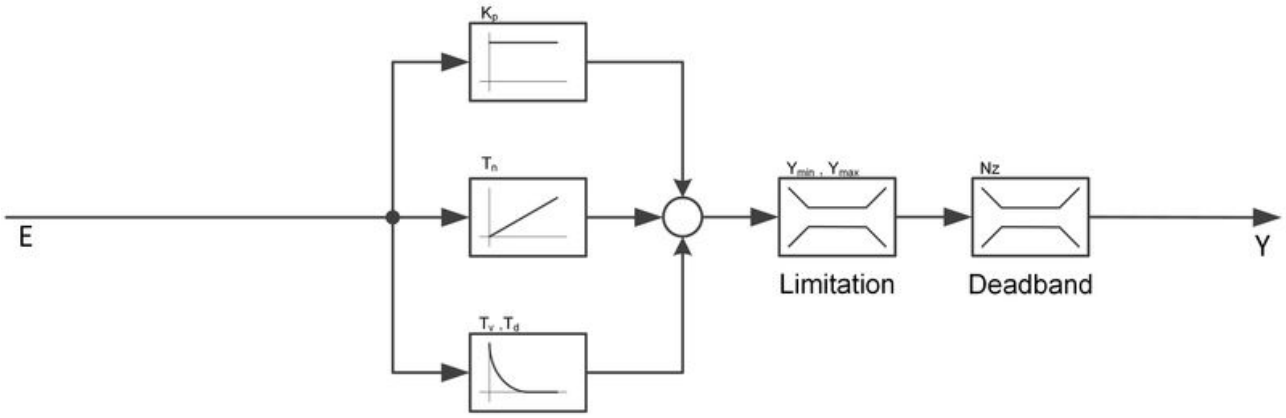
Operation mode "Upstream P part":

(*eOperationMode* = E_BA_PIDMode.eP1ID)



"Parallel structure" operation mode:

(*eOperationMode* = E_BA_PIDMode.ePID)



Control direction

If *bActn* = FALSE, the control direction of the controller is reversed so that a control deviation of less than 0 causes a change in the control value in the positive direction. This is achieved by a negative calculation of the control deviation:

bActn	*fE* (control deviation)	Control direction
TRUE	*fX-fW* (actual value-setpoint)	direct (cooling)
FALSE	*fW-fX* (setpoint-actual value)	indirect (heating)

Passive behavior (bEn = FALSE)

The outputs are set as follows:

fY	0.0
fE	current control deviation, see control direction.

The internal values for the P, I, and D parts are set to 0, also the values for the I and D parts of the preceding cycle. In case of a restart the control value is thus calculated in the first cycle without past values.

Active behavior (bEn = TRUE)

In the first cycle, the I and D parts are calculated without historical values, as already mentioned.

Anti-Reset-Windup

If the I part is active, the controller ensures that it is retained, if the controller output *rY* is about to move beyond the limits *fYMin* or *fYMax*. A preliminary calculation of the controller output takes place inside the controller in every cycle.

Anti-reset windup at min limit

If the precalculation is smaller than the lower output limit *fYMin*, the I part is prevented from falling further and is limited to the value of the last PLC cycle. However, an increase in the I part remains possible.

Anti-reset windup at max limit

On the other hand, if the precalculation is greater than the upper limit *fYMax*, the I part is prevented from increasing further and is also limited to the value of the last PLC cycle. In this case, a drop in the I part remains possible.

Synchronizations

There are several cases where controller output must not only be limited, but also synchronized to a new value by manipulating the I part (if not active, then the D part or the P part). These cases are prioritized because of the potential for simultaneity:

Prio	Description	Conditions	Comments
1	Synchronization via *bSync*/*fSync*	Controller enabled - *bEn* = TRUE	A positive signal on *bSync* sets the I part so that the control value assumes the value *fSync*. If *bEn* and *bSync* are set at the same time, this method can be used to set an initial value from which the controller starts. If the I part is not active, the D part is set accordingly. Note that only the rising edge of *bSync* is evaluated internally as this is a setting action. A TRUE signal must be applied again to the input *bSync* for renewed synchronization, for instance with a transfer value.
2	Range synchronization *fYMin*	Controller enabled - *bEn* = TRUE *fYMin* <> fYMin_1 (last cycle) fY_Test < *fYMin*	If the lower range limit has changed and the precalculated controller output is now smaller than *fYMin* then synchronization is performed to *fYMin*.
3	Range synchronization *fYMax*	Controller enabled - *bEn* = TRUE *fYMax* <> fYMax_1 (last cycle) fY_Test > *fYMax*	if the upper range limit has changed and the precalculated controller output is now greater than *fYMax*, then synchronization is performed to *fYMax*
4	Synchronization with reversal of the control direction	Controller enabled - *bEn* = TRUE *eActn* <> eActn_1 (last cycle)	It is synchronized so that the output holds the value BEFORE the reversal: *fY* = fY_1 (last cycle)
5	Anti-Reset-Windup	Controller enabled - *bEn* = TRUE	see Anti-Reset-Windup

Neutral zone

A value of *fNeutralZone* > 0.0 enables the function of the neutral zone (deadband). A value equal to zero deactivates the deadband element and the values at the input are passed directly through.

If, for the active controller, the change at the input of the element in a PLC cycle is smaller than *fNeutralZone* / 2 in comparison with the previous PLC cycle, then the output is held at the value of the previous cycle until the change is larger than or equal to *fNeutralZone* / 2.

This function is intended to avoid an unnecessarily large number of actuating pulses.

Syntax

```
VAR_INPUT
  bEn                     : BOOL;
  fW                      : REAL;
  fX                      : REAL;
  eActn                   : E_BA_Action := E_BA_Action.eReverse;
  fYMax                   : REAL := 100;
  fYMin                   : REAL := 0;
  bSync                   : BOOL;
  fSync                   : REAL;
END_VAR
VAR_INPUT CONSTANT PERSISTENT
  nCycleCall              : UDINT := 5;
  eOperationMode          : E_BA_PIDMode := E_BA_PIDMode.eP1ID;
  fProportionalConstant   : REAL;
  tIntegralTime           : TIME;
  tDerivativeTime         : TIME;
  tDampingTime            : TIME;
  fNeutralZone            : REAL := 0.0;
END_VAR
VAR_OUTPUT
  fY                      : REAL;
  fE                      : REAL;
END_VAR
```

Inputs

Name	Type	Description
bEn	BOOL	Enable of the controller.
fW	REAL	Setpoint of the controlled system.
fX	REAL	Actual value of the controlled system.
eActn	E_BA_Action [▶ 69]	Control direction of the controller.
fYMax	REAL	Upper output limit of the controller [%].
fYMin	REAL	Lower output limit of the controller [%]. The value *fYMin* is limited upwards by *fYMax*.
bSync	BOOL	Synchronizes the controller to the value of *fSync*.
fSync	REAL	Synchronization value. The value *fSync* is internally limited to values from *fYMin* to *fYMax*.

Inputs CONSTANT PERSISTENT

Name	Type	Description
nCycleCall	UDINT	Call cycle of the function block as a multiple of the cycle time. Internally limited to a minimum value of 1.
eOperationMode	E_BA_PIDMode [▶ 51]	Mode of operation of the controller: PID mode or P-ID mode
fProportionalConstant	REAL	Controller gain. Only affects the P part. Internally limited to a minimum value of 0.
TIntegralTime	TIME	Integral action time of the I part [ms]. A null value at this parameter disables the I part.
tDerivativeTime	TIME	Rate time of the D part [ms]. A null value at this parameter disables the D part.
tDampingTime	TIME	Damping time of the D part [s].
fNeutralZone	REAL	Dead zone

Outputs

Name	Type	Description
fY	REAL	Control value. Range limited by *fYMin* and *fYMax*.

Name	Type	Description
fE	REAL	Control deviation (The calculation depends on the control direction).

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2.2 Universal

4.1.2.2.1 I/O

4.1.2.2.1.1 FB_BA_KL32xx

The function block FB_BA_KL32xx is for the configuration of Bus Terminals of the types KL3208_0010, Kl3201, KL3202 and Kl3204.

Syntax

```
FUNCTION_BLOCK FB_BA_KL32xx
VAR_INPUT
  bConfigurate            : BOOL;
  bReadConfig             : BOOL;
  eSensor                 : E_BA_MeasuringElement;
END_VAR
VAR_OUTPUT
  nState                  : USINT;
  nData                   : INT;
  fVal                    : REAL;
  bErr                    : BOOL;
  bWireBreak              : BOOL;
  bShortCircuit           : BOOL;/
END_VAR
VAR_IN_OUT
  nRawState               : USINT;
  nRawDataIn              : INT;
  nRawCtrl                : USINT;
  nRawDataOut             : INT;
END_VAR
VAR  // [Output-Properties]
  nTerminalType           : WORD;
  nSpecialType            : WORD;
  nFirmwareVersion        : WORD;
  sTerminalDescription    : STRING;
  sSensorName             : STRING;
END_VAR
```

Inputs

Name	Type	Description
bConfigurate	BOOL	A rising edge starts the configuration of the bus terminal.
bReadConfig	BOOL	A rising edge starts the reading of the bus terminal.
eSensor	E_BA_MeasuringElement	Selection of the sensor type.

Outputs

Name	Type	Description
nState	USINT	Output of the present terminal status.
nData	INT	Output of the present process data.
fVal	REAL	Scaled output value.
bErr	BOOL	Error in the terminal configuration.
bWireBreak	BOOL	A TRUE indicates a wire break at the sensor.
bShortCircuit	BOOL	A TRUE indicates a short circuit at the sensor.

Inputs/outputs

Name	Type	Description
nRawState	USINT	Linking with the corresponding status byte of the bus terminal in the I/O area of the program.
nRawDataIn	INT	Linking with the corresponding raw data (Data In) of the bus terminal in the I/O area of the program (0...32767).
nRawCtrl	USINT	Linking with the corresponding control byte of the bus terminal in the I/O area of the program.
nRawDataOut	INT	Linking with the corresponding raw data (Data Out) of the bus terminal in the I/O area of the program.

Properties

Name	Type	Access	Description
FirmwareVersion	WORD	Get	Display of the terminal firmware.
SensorName	STRING	Get	Display of the sensor type.
SpecialType	WORD	Get	Display of the special version of the terminal.
TerminalDescription	STRING	Get	Display of the terminal type and firmware
TerminalType	WORD	Get	Display of the terminal type.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2.2.2 Trigger

4.1.2.2.2.1 FB_BA_ATrigCOV

The function block FB_BA_ATrigCOV is used to detect a value change of a variable *xValue* of any type.

The size of the variable type is internally set to 4 bytes. When the value at *xValue* changes, the output *bQ* is set to TRUE for one cycle, likewise when a rising edge occurs at *bForce*. The block output *bReady* changes to TRUE if the linked variable at *xValue* does not exceed the limit of 4 bytes. If this limit is exceeded, an error message appears in the TwinCAT output window and in the error list. The function block will not check the variable *xValue* any further and will only respond to changes at the input *bForce*. The output *bReady* is then FALSE.

Syntax

```
FUNCTION_BLOCK FB_BA_ATrigCOV
VAR_INPUT
  xValue      : ANY;
  bForce      : BOOL;
END_VAR
VAR_OUTPUT
  bReady      : BOOL;
  bQ          : BOOL;
END_VAR
```

Inputs

Name	Type	Description
xValue	ANY	Value to be monitored. It must not exceed 4 bytes.
bForce	BOOL	A rising edge at this input forces a trigger pulse at the output.

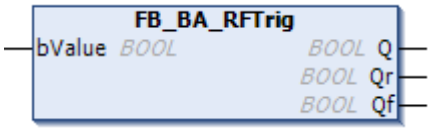
Outputs

Name	Type	Description
bReady	BOOL	Switches to TRUE if the variable applied to *xValue* does not exceed 4 bytes.
bQ	BOOL	In case of a value change at *xValue* or a rising edge at *bForce* this output changes to TRUE for one PLC cycle.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2.2.2.2 FB_BA_RFTrig



The function block FB_BA_RFTrig is used to detect a rising or falling edge on a boolean variable.

Syntax

```
FUNCTION_BLOCK FB_BA_RFTrig
VAR_INPUT
  bValue    :BOOL;
END_VAR
VAR_OUTPUT
  Q         :BOOL;
  Qr        :BOOL;
  Qf        :BOOL;
END_VAR
```

Inputs

Name	Type	Description
bValue	BOOL	Value to be monitored.

Outputs

Name	Type	Description
Q	BOOL	TRUE for one cycle if a rising or falling edge was detected at *bValue*.

Name	Type	Description
Qr	BOOL	TRUE for one cycle if a rising edge was detected at *bValue* (monitored value changes from FALSE to TRUE).
Qf	BOOL	TRUE for one cycle if a falling edge was detected at *bValue* (monitored value changes from TRUE to FALSE).

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2.2.3 Ramps filters

4.1.2.2.3.1 FB_BA_FltrPT1

The FB_BA_FltrPT1 function block serves as a first order filter.

When the function block is first called (system startup), the *fOut* output is automatically set to the *fIn* input once.

Syntax

```
FUNCTION_BLOCK FB_BA_FltrPT1
VAR_INPUT
  fIn            : REAL;
  nDampConst     : UDINT;
  bSetActl       : BOOL;
END_VAR
VAR_OUTPUT
  fOut           : REAL;
END_VAR
```

Inputs

Name	Type	Description
fIn	REAL	Input signal
nDampConst	UDINT	Filter time constant [s]. Internally limited to values between 0 and 86400.
bSetActl	BOOL	A rising edge at this input switches the output value *fOut* to the input value *fIn*.

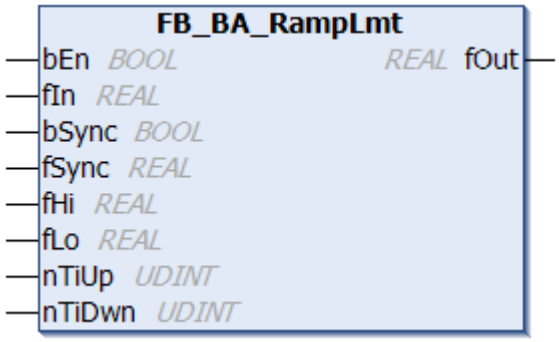
Outputs

Name	Type	Description
fOut	REAL	Filtered output signal

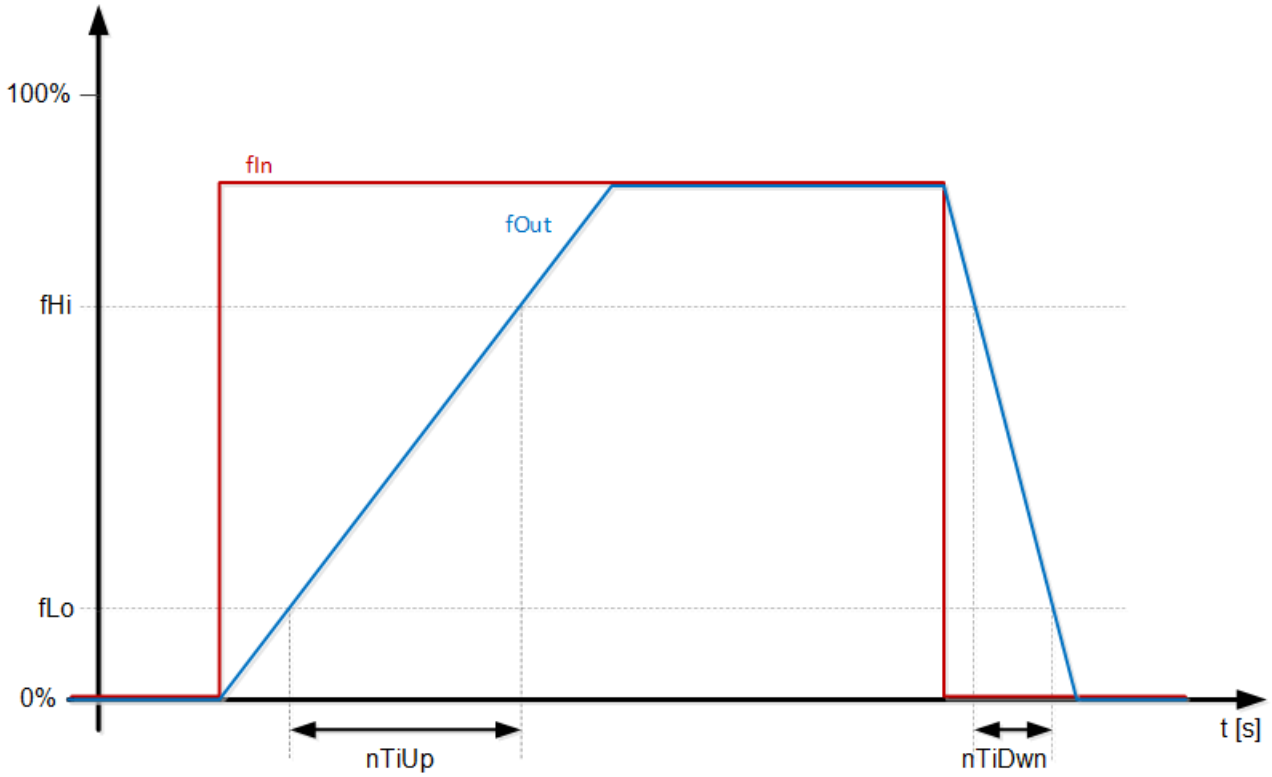
Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.1.2.2.3.2 FB_BA_RampLmt



The function block FB_BA_RampLmt limits the increase or decrease speed of an input signal.
When *fIn* rises, the output *fOut* is limited to the slope of (*fHi-fLo*)/*nTiUp*.
When *fIn* falls, the output *fOut* is limited to the slope of (*fHi-fLo*)/*nTiDwn*.



Syntax

```
FUNCTION_BLOCK FB_BA_RampLmt
VAR_INPUT
  bEn             : REAL;
  fIn             : REAL;
  bSync           : BOOL;
  fSync           : REAL;
  fHi             : REAL;
  fLo             : REAL;
  nTiUp           : UDINT;
  nTiDwn          : UDINT;
END_VAR
VAR_OUTPUT
  fOut            : REAL;
END_VAR
```

Inputs

Name	Type	Description
bEn	BOOL	Enable function block if FALSE, then *fOut* = 0.0.

Name	Type	Description
bEnRamp	BOOL	Enable ramp limitation, if FALSE, then *fOut* = *fIn*.
fIn	REAL	Input value of the ramp function
fHi	REAL	Upper interpolation point for calculating the ramps
fLo	REAL	Lower interpolation point for calculating the ramps. *fHi* must be greater than *fLo*, otherwise an error is output!
nTiUp	UDINT	Rise time [s]
nTiDwn	UDINT	Fall time [s]

Outputs

Name	Type	Description
fOut	REAL	Output signal, slope-limited through the ramps.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2 DUTs

4.2.1 Enumerations

4.2.1.1 [Functional]

4.2.1.1.1 E_BA_CompareMode

The enumeration type variable is used to display and classify comparison results.

Syntax

```
TYPE E_BA_CompareMode :
(
  Invalid           := 0,

  eLower            := 1,
  eLowerOrEqual     := 2,
  eEqual            := 3,
  eNotEqual         := 4,
  eHigherOrEqual    := 5,
  eHigher           := 6

) BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eLower	The comparison value is smaller.
eLowerOrEqual	The comparison value is less than or equal to.
eEqual	The comparison value is the same.
eNotEqual	The comparison value is not the same.
eHigherOrEqual	The comparison value is greater than or equal to.
eHigher	The comparison value is larger.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.2 Controller

4.2.1.2.1 E_BA_PIDMode

Selection of the controller structure.

Syntax

```
TYPE E_BA_PIDMode:
(
  Invalid         := 0,
  eP1ID           := 1,
  ePID            := 2
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eP1ID	P element is upstream.
ePID	P-, I- and D-element in parallel structure.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.3 Conversion

4.2.1.3.1 E_BA_ByteMappingMode

```
TYPE E_BA_ByteMappingMode :
(
  Invalid             := 0,
  eIndex1N            := 1,
  eBinary1N           := 2,
  eIndexUpDown        := 3,
  eBinaryUpDown       := 4,
  eBinaryDecimal      := 5,
) BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eIndex1N	Sets the indexed bit to TRUE. Sample: 2#0000_0001 | 1 2#0000_0010 | 2 2#0000_0100 | 3 2#0000_1000 | 4
eBinary1N	Sets only the first bit of a binary mapped decimal value to TRUE. Sample: 2#0000_0001 | 1 2#0000_0010 | 2, 3 2#0000_0100 | 4, 5, 6

Name	Description
	2#0000_1000 | 8, 9, 10, 11, 12, 13, 14, 15
eIndexUpDown	Sets a specific number of bits to TRUE. Sample: 2#0000_0001 | 1 2#0000_0011 | 2 2#0000_0111 | 3 2#0000_1111 | 4
eBinaryUpDown	Sets all bits of a binary mapped decimal value to TRUE. Sample: 2#0000_0001 | 1 2#0000_0011 | 2, 3 2#0000_0111 | 4, 5, 6 2#0000_1111 | 8, 9, 10, 11, 12, 13, 14, 15
eBinaryDecimal	Binary mapped decimal value. Sample: 2#0000_0001 | 1 2#0000_0010 | 2 2#0000_0011 | 3 2#0000_0100 | 4 2#0000_0101 | 5 2#1111_1111 | 255

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.4 Event

4.2.1.4.1 E_BA_EventState

The enumeration is used to display and classify events.

Syntax

```
TYPE E_BA_EventState:
(
  Invalid        := 0,
  Unknown        := 1,

  eNormal        := 2,
  eFault         := 3,
  eOffNormal     := 4,
  eLowLimit      := 5;
  eHighLimit     := 6
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user
eUnknown	State unknown
eNormal	Normal operation
eFault	Invalid value
eOffnormal	Alarm state

Name	Description
eLowLimit	Lower limit value undershot
eHighLimit	Upper limit value exceeded

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.4.2 E_BA_EventTransition

Describes the possible transition states of event objects.

Syntax

```
TYPE E_BA_EventTransition:
(
  Invalid        := 0,

  eToOffnormal   := 1,
  eToFault       := 2,
  eToNormal      := 3
);
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eToOffnormal	Transition to *Alarm* state.
eToFault	Transition to *Error* state.
eToNormal	Transition to *Normal* state.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5 Types

4.2.1.5.1 Date and Time

4.2.1.5.1.1 E_BA_DateValChoice

The enumeration is used to select time periods in schedules.

Syntax

```
TYPE E_BA_DateValChoice:
(
  Invalid        := 0,
  eDate          := 1,
  eDateRange     := 2,
  eWeekDay       := 3
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eDate	Selection of a single date.
eDateRange	Selection of a date range.
eWeekNDay	Selection of a day of the week.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.1.2 E_BA_Day

The enumeration is used to specify days within a month.

Syntax

```
TYPE E_BA_Day:
(
  Invalid           := 0,
  Unspecified       := 16#FF,

  eDay01            := 1,
  eDay02            := 2,
  eDay03            := 3,
  eDay04            := 4,
  eDay05            := 5,
  eDay06            := 6,
  eDay07            := 7,
  eDay08            := 8,
  eDay09            := 9,
  eDay10            := 11,
  eDay11            := 11,
  eDay12            := 12,
  eDay13            := 13,
  eDay14            := 14,
  eDay15            := 15,
  eDay16            := 16,
  eDay17            := 17,
  eDay18            := 18,
  eDay19            := 19,
  eDay20            := 20,
  eDay21            := 21,
  eDay22            := 22,
  eDay23            := 23,
  eDay24            := 24,
  eDay25            := 25,
  eDay26            := 26,
  eDay27            := 27,
  eDay28            := 28,
  eDay29            := 29,
  eDay30            := 30,
  eDay31            := 31,

  eLastDayOfMonth   := 32,
  eOddDaysOfMonth   := 33,
  eEvenDaysOfMonth  := 34
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
Unspecified	Not specified.
eDayNN	Day no. NN (1…31) of the month is specified.
eLastDayOfMonth	The last day of the month is specified.
eOddDaysOfMonth	The odd days of the month are specified.
eEvenDaysOfMonth	The even days of the month are specified.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.1.3 E_BA_Month

The enumeration is used to specify months.

Syntax

```
TYPE E_BA_Month:
(
  Invalid         := 0,
  Unspecified     := 16#FF,

  eJanuary        := 1,
  eFebruary       := 2,
  eMarch          := 3,
  eApril          := 4,
  eMay            := 5,
  eJune           := 6,
  eJuly           := 7,
  eAugust         := 8,
  eSeptember      := 9,
  eOctober        := 10,
  eNovember       := 11,
  eDecember       := 12,

  eOddMonths      := 13,
  eEvenMonths     := 14
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
Unspecified	Not specified.
eJanuary	January is specified.
eFebruary	February is specified.
eMarch	March is specified.
eApril	April is specified.
eMay	May is specified.
eJune	June is specified.
eJuly	July is specified.
eAugust	August is specified.
eSeptember	September is specified.
eOctober	October is specified.
eNovember	November is specified.
eDecember	December is specified.
eOddMonths	The months with odd ordinal number are specified.
eEvenMonths	The months with even ordinal number are specified.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.1.4 E_BA_Week

The enumeration is used to specify weeks within a month.

Syntax

```
TYPE E_BA_Week:
(
  Invalid         := 0,
  Unspecified     := 16#FF,

  eWeek1          := 1,
  eWeek2          := 2,
```

```
  eWeek3          := 3,
  eWeek4          := 4,
  eWeek5          := 5
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
Unspecified	Not specified.
eWeek1	The first week within the month is specified.
eWeek2	The second week within the month is specified.
eWeek3	The third week within the month is specified.
eWeek4	The fourth week within the month is specified.
eWeek5	The fifth week within the month is specified.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.1.5 E_BA_Weekday

The enumeration is used to specify days of the week.

Syntax

```
TYPE E_BA_Weekday:
(
  Invalid         := 0,
  Unspecified     := 16#FF,

  eMonday         := 1,
  eTuesday        := 2,
  eWednesday      := 3,
  eThursday       := 4,
  eFriday         := 5,
  eSaturday       := 6,
  eSunday         := 7
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
Unspecified	Not specified.
eMonday	Monday is specified.
eTuesday	Tuesday is specified.
eWednesday	Wednesday is specified.
eThursday	Thursday is specified.
eFriday	Friday is specified.
eSaturday	Saturday is specified.
eSunday	Sunday is specified.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.2 E_BA_DataClass

The enumeration is used to specify values.

Syntax

```
TYPE E_BA_DataClass:
(
  Invalid        := 0,
  Unknown        := 1,
  Null           := 2,

  eAnalog        := 10,
  eBinary        := 11,
  eMultistate    := 12
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
Unknown	Unknown.
Zero	Not specified / not set.
eAnalog	Analog object.
eBinary	Binary object.
eMultistate	Multistate object.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.3 E_BA_DataType

The enumeration is used to specify PLC data types.

Syntax

```
TYPE E_BA_DataType :
(
  Invalid       := 0,
  Undefined     := 1,

  eBool         := 10,
  eBit          := 11,
  eByte         := 12,
  eWord         := 13,
  eDWord        := 14,
  eLWord        := 15,
  eSInt         := 16,
  eInt          := 17,
  eDInt         := 18,
  eLInt         := 19,
  eUSInt        := 20,
  eUInt         := 21,
  eUDInt        := 22,
  eULInt        := 23,
  eReal         := 24,
  eLReal        := 25,
  eString       := 26,
  eWString      := 27,
  eTime         := 28,
  eDate         := 29,
  eDateTime     := 30,
  eTimeOfDay    := 31,

  eEnumeration  := 32,

  eStructure    := 33,
  eInterface    := 34,
) BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.

Name	Description
Undefined	Not specified / not set.
eBool	Variable of the data type BOOL.
eBit	Variable of the data type BIT.
eByte	Variable of the data type BYTE.
eWord	Variable of the data type WORD.
eDWord	Variable of the data type DWORD.
eLWord	Variable of the data type LWORD.
eSInt	Variable of the data type SINT.
eInt	Variable of the data type INT.
eDInt	Variable of the data type DINT.
eLInt	Variable of the data type LINT.
eUSInt	Variable of the data type USINT.
eUInt	Variable of the data type UINT.
eUDInt	Variable of the data type UDINT.
eULInt	Variable of the data type ULINT.
eReal	Variable of the data type REAL.
eLReal	Variable of the data type LREAL.
eString	Variable of the data type STRING.
eWString	Variable of the data type WSTRING.
eTime	Variable of the data type TIME
eDate	Variable of the data type DATE
eDateTime	Variable of the data type DATE_AND_TIME or DT.
eTimeOfDay	Variable of the data type TIME_OF_DAY or TOD.
eEnumeration	Variable of the data type ENUMERATION.
eStructure	Variable of the data type STRUCT.
eInterface	INTERFACE data type.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.4 Schedule

4.2.1.5.4.1 E_BA_SchedEntryState

The enumeration is used to specify entries in schedules.

Syntax

```
TYPE E_BA_SchedEntryState:
(
  Invalid        := 0,

  eUndefined     := 1,
  eValue         := 2,
  eNull          := 3
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eUndefined	Not specified.
eValue	A value is set.

Name	Description
eNull	Not specified / not set.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.5 Trend

4.2.1.5.5.1 E_BA_LoggingType

The enumeration is used to classify the storage of trend data.

Syntax

```
TYPE E_BA_LoggingType:
(
  Invalid        := -1,

  ePolled        := 0,
  eCOV           := 1,
  eTriggered     := 2
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
ePolled	Data is retrieved and stored cyclically.
eCOV	Data is saved when changed (plus hysteresis).
eTriggered	Data is stored after the call.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.5.5.2 E_BA_TrendEntryType

The enumeration specifies the type of the trend entry. It is the extension of E_BA_DataClass [▶ 56].

Syntax

```
TYPE E_BA_TrendEntryType:
(
  Invalid        := E_BA_DataClass.Invalid,

  eBinary        := E_BA_DataClass.eBinary,
  eAnalog        := E_BA_DataClass.eAnalog,
  eMultistate    := E_BA_DataClass.eMultistate,

  eEvent         := 20
)BYTE;
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eBinary	Entry with a binary value.
eAnalog	Entry with an analog value.
eMultistate	Entry with a multistate value.
eEvent	Event entry.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.6 Units

4.2.1.6.1 E_BA_Unit

The enumeration is used to specify units.

Syntax

```
TYPE E_BA_Unit:
(
  Invalid                                 := -1,

  eArea_SquareMeters                      := 0,
  eArea_SquareFeet                        := 1,
  eElectrical_Milliamperes                := 2,
  eElectrical_Amperes                     := 3,
  eElectrical_Ohms                        := 4,
  eElectrical_Volts                       := 5,
  eElectrical_Kilovolts                   := 6,
  eElectrical_Megavolts                   := 7,
  eElectrical_VoltAmperes                 := 8,
  eElectrical_KilovoltAmperes             := 9,
  eElectrical_MegavoltAmperes             := 10,
  eElectrical_VoltAmperesReactive         := 11,
  eElectrical_KilovoltAmperesReactive     := 12,
  eElectrical_MegavoltAmperesReactive     := 13,
  eElectrical_DegreesPhase                := 14,
  eElectrical_PowerFactor                 := 15,
  eEnergy_Joules                          := 16,
  eEnergy_Kilojoules                      := 17,
  eEnergy_WattHours                       := 18,
  eEnergy_KilowattHours                   := 19,
  eEnergy_Btus                            := 20,
  eEnergy_Therms                          := 21,
  eEnergy_TonHours                        := 22,
  eEnthalpy_JoulesPerKilogramDryAir       := 23,
  eEnthalpy_BtusPerPoundDryAir            := 24,
  eFrequency_CyclesPerHour                := 25,
  eFrequency_CyclesPerMinute              := 26,
  eFrequency_Hertz                        := 27,
  eHumidity_GramsOfWaterPerKilogramDryAir := 28,
  eHumidity_PercentRelativeHumidity       := 29,
  eLength_Millimeters                     := 30,
  eLength_Meters                          := 31,
  eLength_Inches                          := 32,
  eLength_Feet                            := 33,
  eLight_WattsPerSquareFoot               := 34,
  eLight_WattsPerSquareMeter              := 35,
  eLight_Lumens                           := 36,
  eLight_Luxes                            := 37,
  eLight_FootCandles                      := 38,
  eMass_Kilograms                         := 39,
  eMass_PoundsMass                        := 40,
  eMass_Tons                              := 41,
  eMassFlow_KilogramsPerSecond            := 42,
  eMassFlow_KilogramsPerMinute            := 43,
  eMassFlow_KilogramsPerHour              := 44,
  eMassFlow_PoundsMassPerMinute           := 45,
  eMassFlow_PoundsMassPerHour             := 46,
  ePower_Watts                            := 47,
  ePower_Kilowatts                        := 48,
  ePower_Megawatts                        := 49,
  ePower_BtusPerHour                      := 50,
  ePower_Horsepower                       := 51,
  ePower_TonsRefrigeration                := 52,
  ePressure_Pascals                       := 53,
  ePressure_Kilopascals                   := 54,
  ePressure_Bars                          := 55,
  ePressure_PoundsForcePerSquareInch      := 56,
  ePressure_CentimetersOfWater            := 57,
```

```
ePressure_InchesOfWater                    := 58,
ePressure_MillimetersOfMercury             := 59,
ePressure_CentimetersOfMercury             := 60,
ePressure_InchesOfMercury                  := 61,
eTemperature_DegreesCelsius                := 62,
eTemperature_DegreesKelvin                 := 63,
eTemperature_DegreesFahrenheit             := 64,
eTemperature_DegreeDaysCelsius             := 65,
eTemperature_DegreeDaysFahrenheit          := 66,
eTime_Years                                := 67,
eTime_Months                               := 68,
eTime_Weeks                                := 69,
eTime_Days                                 := 70,
eTime_Hours                                := 71,
eTime_Minutes                              := 72,
eTime_Seconds                              := 73,
eVelocity_MetersPerSecond                  := 74,
eVelocity_KilometersPerHour                := 75,
eVelocity_FeetPerSecond                    := 76,
eVelocity_FeetPerMinute                    := 77,
eVelocity_MilesPerHour                     := 78,
eVolume_CubicFeet                          := 79,
eVolume_CubicMeters                        := 80,
eVolume_ImperialGallons                    := 81,
eVolume_Liters                             := 82,
eVolume_UsGallons                          := 83,
eVolumetricFlow_CubicFeetPerMinute         := 84,
eVolumetricFlow_CubicMetersPerSecond       := 85,
eVolumetricFlow_ImperialGallonsPerMinute   := 86,
eVolumetricFlow_LitersPerSecond            := 87,
eVolumetricFlow_LitersPerMinute            := 88,
eVolumetricFlow_UsGallonsPerMinute         := 89,
eOther_DegreesAngular                      := 90,
eOther_DegreesCelsiusPerHour               := 91,
eOther_DegreesCelsiusPerMinute             := 92,
eOther_DegreesFahrenheitPerHour            := 93,
eOther_DegreesFahrenheitPerMinute          := 94,
eOther_NoUnits                             := 95,
eOther_PartsPerMillion                     := 96,
eOther_PartsPerBillion                     := 97,
eOther_Percent                             := 98,
eOther_PercentPerSecond                    := 99,
eOther_PerMinute                           := 100,
eOther_PerSecond                           := 101,
eOther_PsiPerDegreeFahrenheit              := 102,
eOther_Radians                             := 103,
eOther_RevolutionsPerMinute                := 104,
eCurrency_Currency1                        := 105,
eCurrency_Currency2                        := 106,
eCurrency_Currency3                        := 107,
eCurrency_Currency4                        := 108,
eCurrency_Currency5                        := 109,
eCurrency_Currency6                        := 110,
eCurrency_Currency7                        := 111,
eCurrency_Currency8                        := 112,
eCurrency_Currency9                        := 113,
eCurrency_Currency10                       := 114,
eArea_SquareInches                         := 115,
eArea_SquareCentimeters                    := 116,
eEnthalpy_BtusPerPound                     := 117,
eLength_Centimeters                        := 118,
eMassFlow_PoundsMassPerSecond              := 119,
eTemperature_DeltaDegreesFahrenheit        := 120,
eTemperature_DeltaDegreesKelvin            := 121,
eElectrical_Kilohms                        := 122,
eElectrical_Megohms                        := 123,
eElectrical_Millivolts                     := 124,
eEnergy_KilojoulesPerKilogram              := 125,
eEnergy_Megajoules                         := 126,
eEntropy_JoulesPerDegreeKelvin             := 127,
eEntropy_JoulesPerKilogramDegreeKelvin     := 128,
eFrequency_Kilohertz                       := 129,
eFrequency_Megahertz                       := 130,
eFrequency_PerHour                         := 131,
ePower_Milliwatts                          := 132,
ePressure_Hectopascals                     := 133,
ePressure_Millibars                        := 134,
eVolumetricFlow_CubicMetersPerHour         := 135,
eVolumetricFlow_LitersPerHour              := 136,
eOther_KilowattHoursPerSquareMeter         := 137,
```

```
eOther_KilowattHoursPerSquareFoot          := 138,
eOther_MegajoulesPerSquareMeter            := 139,
eOther_MegajoulesPerSquareFoot             := 140,
eOther_WattsPerSquareMeterDegreeKelvin     := 141,
eVolumetricFlow_CubicFeetPerSecond         := 142,
eOther_PercentObscurationPerFoot           := 143,
eOther_PercentObscurationPerMeter          := 144,
eElectrical_Milliohms                      := 145,
eEnergy_MegawattHours                      := 146,
eEnergy_KiloBtus                           := 147,
eEnergy_MegaBtus                           := 148,
eEnthalpy_KilojoulesPerKilogramDryAir      := 149,
eEnthalpy_MegajoulesPerKilogramDryAir      := 150,
eEntropy_KilojoulesPerDegreeKelvin         := 151,
eEntropy_MegajoulesPerDegreeKelvin         := 152,
eForce_Newton                              := 153,
eMassFlow_GramsPerSecond                   := 154,
eMassFlow_GramsPerMinute                   := 155,
eMassFlow_TonsPerHour                      := 156,
ePower_KiloBtusPerHour                     := 157,
eTime_HundredthsSeconds                    := 158,
eTime_Milliseconds                         := 159,
eTorque_NewtonMeters                       := 160,
eVelocity_MillimetersPerSecond             := 161,
eVelocity_MillimetersPerMinute             := 162,
eVelocity_MetersPerMinute                  := 163,
eVelocity_MetersPerHour                    := 164,
eVolumetricFlow_CubicMetersPerMinute       := 165,
eAcceleration_MetersPerSecondPerSecond     := 166,
eElectrical_AmperesPerMeter                := 167,
eElectrical_AmperesPerSquareMeter          := 168,
eElectrical_AmpereSquareMeters             := 169,
eElectrical_Farads                         := 170,
eElectrical_Henrys                         := 171,
eElectrical_OhmMeters                      := 172,
eElectrical_Siemens                        := 173,
eElectrical_SiemensPerMeter                := 174,
eElectrical_Teslas                         := 175,
eElectrical_VoltsPerDegreeKelvin           := 176,
eElectrical_VoltsPerMeter                  := 177,
eElectrical_Webers                         := 178,
eLight_Candelas                            := 179,
eLight_CandelasPerSquareMeter              := 180,
eTemperature_DegreesKelvinPerHour          := 181,
eTemperature_DegreesKelvinPerMinute        := 182,
eOther_JouleSeconds                        := 183,
eOther_RadiansPerSecond                    := 184,
eOther_SquareMetersPerNewton               := 185,
eOther_KilogramsPerCubicMeter              := 186,
eOther_NewtonSeconds                       := 187,
eOther_NewtonsPerMeter                     := 188,
eOther_WattsPerMeterPerDegreeKelvin        := 189,
eMicro_Siemens                             := 190,
eCubic_FeetPerHour                         := 191,
eUs_GallonsPerHour                         := 192,
eKilometers                                := 193,
eMicrometers                               := 194,
eGrams                                     := 195,
eMilligrams                                := 196,
eMilliliters                               := 197,
eMillilitersPerSecond                      := 198,
eDecibels                                  := 199,
eDecibelsMillivolt                         := 200,
eDecibelsVolt                              := 201,
eMillisiemens                              := 202,
eWatt_HoursReactive                        := 203,
eKilowattHoursReactive                     := 204,
eMegawattHoursReactive                     := 205,
eMillimetersOfWater                        := 206,
ePer_Mille                                 := 207,
eGrams_PerGram                             := 208,
eKilograms_PerKilogram                     := 209,
eGrams_PerKilogram                         := 210,
eMilligrams_PeGram                         := 211,
eMilligrams_PeKilogram                     := 212,
eGrams_PerMilliliter                       := 213,
eGrams_PerLiter                            := 214,
eMilligrams_PerLiter                       := 215,
eMicrograms_PerLiter                       := 216,
eGrams_PerCubicMeter                       := 217,
```

```
  eMilligrams_PerCubicMeter                 := 218,
  eMicrograms_PerCubicMeter                 := 219,
  eNanograms_PerCubicMeter                  := 220,
  eGrams_PerCubicCentimeter                 := 221,
  eBecquerels                               := 222,
  eKilobecquerels                           := 223,
  eMegabecquerels                           := 224,
  eGray                                     := 225,
  eMilligray                                := 226,
  eMicrogray                                := 227,
  eSieverts                                 := 228,
  eMillisieverts                            := 229,
  eMicrosieverts                            := 230,
  eMicrosievertsPerHour                     := 231,
  eDecibels_A                               := 232,
  eNephelometric_TurbidityUnit              := 233,
  ePH                                       := 234,
  eGrams_PerSquareMeter                     := 235,
  eMinutes_PerDegreeKelvin                  := 236
)INT;
End_TYPE
```

Name	Description
Invalid	No significance for the user.
eArea_SquareMeters	Square meters
eArea_SquareFeet	Square feet
eElectrical_Milliamperes	Milliamperes
eElectrical_Amperes	Amperes
eElectrical_Ohms	Ohms
eElectrical_Volts	Volts
eElectrical_Kilovolts	Kilovolts
eElectrical_Megavolts	Megavolts
eElectrical_VoltAmperes	Volt-amperes
eElectrical_KilovoltAmperes	Kilovolt-amperes
eElectrical_MegavoltAmperes	Megavolt-amperes
eElectrical_VoltAmperesReactive	Volt-amperes reactive
eElectrical_KilovoltAmperesReactive	Kilovolt-amperes reactive
eElectrical_MegavoltAmperesReactive	Megavolt-amperes reactive
eElectrical_DegreesPhase	Phase position in degrees
eElectrical_PowerFactor	Power factor cos φ
eEnergy_Joules	Joules
eEnergy_Kilojoules	Kilojoules
eEnergy_WattHours	Watt-hours
eEnergy_KilowattHours	Kilowatt-hours
eEnergy_Btus	BTUs (British Thermal Unit)
eEnergy_Therms	Therms
eEnergy_TonHours	Ton-hours
eEnthalpy_JoulesPerKilogramDryAir	Joules per kg dry air
eEnthalpy_BtusPerPoundDryAir	BTUs per pound dry air
eFrequency_CyclesPerHour	Cycles per hour

Name	Description
eFrequency_CyclesPerMinute	Cycles per minute
eFrequency_Hertz	Hertz
eHumidity_GramsOfWaterPerKilogramDryAir	Grams of water per kilogram dry air
eHumidity_PercentRelativeHumidity	Percent relative humidity
eLength_Millimeters	Millimeters
eLength_Meters	Meters
eLength_Inches	Inches
eLength_Feet	Feet
eLight_WattsPerSquareFoot	Watts per square foot
eLight_WattsPerSquareMeter	Watts per square meter
eLight_Lumens	Lumens
eLight_Luxes	Luxes
eLight_FootCandles	Foot candles (Anglo-Saxon unit of measurement for illuminance)
eMass_Kilograms	Kilograms
eMass_PoundsMass	Pounds mass
eMass_Tons	Tons
eMassFlow_KilogramsPerSecond	Kilograms per second
eMassFlow_KilogramsPerMinute	Kilograms per minute
eMassFlow_KilogramsPerHour	Kilograms per hour
eMassFlow_PoundsMassPerMinute	Pounds mass per minute
eMassFlow_PoundsMassPerHour	Pounds mass per hour
ePower_Watts	Watts
ePower_Kilowatts	Kilowatts
ePower_Megawatts	Megawatts
ePower_BtusPerHour	BTUs per hour
ePower_Horsepower	Horsepower
ePower_TonsRefrigeration	Cooling capacity - energy to melt a ton of ice in 24h.
ePressure_Pascals	Pascals
ePressure_Kilopascals	Kilopascals
ePressure_Bars	Bars
ePressure_PoundsForcePerSquareInch	Pounds force per square foot
ePressure_CentimetersOfWater	Centimeters of water
ePressure_InchesOfWater	Inches of water
ePressure_MillimetersOfMercury	Millimeters of mercury
ePressure_CentimetersOfMercury	Centimeters of mercury

Name	Description
ePressure_InchesOfMercury	Inches of mercury
eTemperature_DegreesCelsius	Degrees Celsius
eTemperature_DegreesKelvin	Degrees Kelvin
eTemperature_DegreesFahrenheit	Degrees Fahrenheit
eTemperature_DegreeDaysCelsius	Degree days Celsius
eTemperature_DegreeDaysFahrenheit	Degree days Fahrenheit
eTime_Years	Years
eTime_Months	Months
eTime_Weeks	Weeks
eTime_Days	Days
eTime_Hours	Hours
eTime_Minutes	Minutes
eTime_Seconds	Seconds
eVelocity_MetersPerSecond	Meters per second
eVelocity_KilometersPerHour	Kilometers per hour
eVelocity_FeetPerSecond	Feet per second
eVelocity_FeetPerMinute	Feet per hour
eVelocity_MilesPerHour	Miles per hour
eVolume_CubicFeet	Cubic feet
eVolume_CubicMeters	Cubic meters
eVolume_ImperialGallons	Imperial gallons
eVolume_Liters	Liters
eVolume_UsGallons	US gallons
eVolumetricFlow_CubicFeetPerMinute	Cubic feet per minute
eVolumetricFlow_CubicMetersPerSecond	Cubic meters per second
eVolumetricFlow_ImperialGallonsPerMinute	Imperial gallons per minute
eVolumetricFlow_LitersPerSecond	Liters per second
eVolumetricFlow_LitersPerMinute	Liters per minute
eVolumetricFlow_UsGallonsPerMinute	US gallons per minute
eOther_DegreesAngular	Degrees Angular
eOther_DegreesCelsiusPerHour	Degrees Celsius per hour
eOther_DegreesCelsiusPerMinute	Degrees Celsius per minute
eOther_DegreesFahrenheitPerHour	Degrees Fahrenheit per hour
eOther_DegreesFahrenheitPerMinute	Degrees Fahrenheit per minute

Name	Description
eOther_NoUnits	No units
eOther_PartsPerMillion	Parts per million
eOther_PartsPerBillion	Parts per billion
eOther_Percent	Percent
eOther_PercentPerSecond	Percent per second
eOther_PerMinute	Per minute
eOther_PerSecond	Per second
eOther_PsiPerDegreeFahrenheit	Psi per degree Fahrenheit
eOther_Radians	Radians
eOther_RevolutionsPerMinute	Revolutions per minute
eCurrency_CurrencyN	Currency
eArea_SquareInches	Square inches
eArea_SquareCentimeters	Square centimeters
eEnthalpy_BtusPerPound	BTUs per pound
eLength_Centimeters	Centimeters
eMassFlow_PoundsMassPerSecond	Pounds mass per second
eTemperature_DeltaDegreesFahrenheit	Delta degrees Fahrenheit
eTemperature_DeltaDegreesKelvin	Delta degrees Kelvin
eElectrical_Kilohms	Kilohms
eElectrical_Megohms	Megohms
eElectrical_Millivolts	Millivolts
eEnergy_KilojoulesPerKilogram	Kilojoules
eEnergy_Megajoules	Megajoules
eEntropy_JoulesPerDegreeKelvin	Joules per degree Kelvin
eEntropy_JoulesPerKilogramDegreeKelvin	Joules per kilogram degree Kelvin
eFrequency_Kilohertz	Kilohertz
eFrequency_Megahertz	Megahertz
eFrequency_PerHour	Per hour
ePower_Milliwatts	Milliwatts
ePressure_Hectopascals	Hectopascals
ePressure_Millibars	Millibars
eVolumetricFlow_CubicMetersPerHour	Cubic meters per hour
eVolumetricFlow_LitersPerHour	Liters per hour
eOther_KilowattHoursPerSquareMeter	Kilowatt hours per square meter
eOther_KilowattHoursPerSquareFoot	Kilowatt hours per square foot
eOther_MegajoulesPerSquareMeter	Megajoules per square meter

Name	Description
eOther_MegajoulesPerSquareFoot	Megajoules per square foot
eOther_WattsPerSquareMeterDegreeKelvin	Watts per square meter degree Kelvin
eVolumetricFlow_CubicFeetPerSecond	Cubic feet per second
eOther_PercentObscurationPerFoot	Percent obscuration per foot
eOther_PercentObscurationPerMeter	Percent obscuration per meter
eElectrical_Milliohms	Milliohms
eEnergy_MegawattHours	Megawatt hours
eEnergy_KiloBtus	Kilo BTUs
eEnergy_MegaBtus	Mega BTUs
eEnthalpy_KilojoulesPerKilogramDryAir	Kilojoules per kilogram dry air
eEnthalpy_MegajoulesPerKilogramDryAir	Megajoules per kilogram dry air
eEntropy_KilojoulesPerDegreeKelvin	Kilojoules per degree Kelvin
eEntropy_MegajoulesPerDegreeKelvin	Megajoules per degree Kelvin
eForce_Newton	Newtons
eMassFlow_GramsPerSecond	Grams per second
eMassFlow_GramsPerMinute	Grams per minute
eMassFlow_TonsPerHour	Tons per hour
ePower_KiloBtusPerHour	Kilo BTUs per hour
eTime_HundredthsSeconds	Hundredths seconds
eTime_Milliseconds	Milliseconds
eTorque_NewtonMeters	Newton meters
eVelocity_MillimetersPerSecond	Millimeters per second
eVelocity_MillimetersPerMinute	Millimeters per minute
eVelocity_MetersPerMinute	Meters per minute
eVelocity_MetersPerHour	Meters per hour
eVolumetricFlow_CubicMetersPerMinute	Cubic meters per minute
eAcceleration_MetersPerSecondPerSecond	Meters per second per second
eElectrical_AmperesPerMeter	Amperes per meter
eElectrical_AmperesPerSquareMeter	Amperes per square meter
eElectrical_AmpereSquareMeters	Ampere square meters
eElectrical_Farads	Farads
eElectrical_Henrys	Henrys
eElectrical_OhmMeters	Ohm-meters

Name	Description
eElectrical_Siemens	Siemens
eElectrical_SiemensPerMeter	Siemens per meter
eElectrical_Teslas	Teslas
eElectrical_VoltsPerDegreeKelvin	Volts per degree Kelvin
eElectrical_VoltsPerMeter	Volts per meter
eElectrical_Webers	Webers
eLight_Candelas	Candelas
eLight_CandelasPerSquareMeter	Candelas per square meter
eTemperature_DegreesKelvinPerHour	Degrees Kelvin per hour
eTemperature_DegreesKelvinPerMinute	Degrees Kelvin per minute
eOther_JouleSeconds	Joule-seconds (angular momentum)
eOther_RadiansPerSecond	Radians per second
eOther_SquareMetersPerNewton	Square meters per Newton
eOther_KilogramsPerCubicMeter	Kilograms per cubic meter
eOther_NewtonSeconds	Newton-seconds (impulse)
eOther_NewtonsPerMeter	Newtons per meter
eOther_WattsPerMeterPerDegreeKelvin	Watts per meter per degree Kelvin
eMicro_Siemens	Microsiemens
eCubic_FeetPerHour	Cubic feet per hour
eUs_GallonsPerHour	US gallons per hour
eKilometers	Kilometers
eMicrometers	Micrometers
eGrams	Grams
eMilligrams	Milligrams
eMilliliters	Milliliters
eMillilitersPerSecond	Milliliters per second
eDecibels	Decibels
eDecibelsMillivolt	Decibels millivolt
eDecibelsVolt	Decibels Volt
eMillisiemens	Millisiemens
eWatt_HoursReactive	Watt-hours reactive
eKilowattHoursReactive	Kilowatt-hours reactive
eMegawattHoursReactive	Megawatt-hours reactive
eMillimetersOfWater	Millimeters of water
ePer_Mille	Per mille
eGrams_PerGram	Grams per gram
eKilograms_PerKilogram	Kilograms per kilogram
eGrams_PerKilogram	Grams per kilogram
eMilligrams_PeGram	Milligrams per gram
eMilligrams_PeKilogram	Milligrams per kilogram
eGrams_PerMilliliter	Grams per milliliter

Name	Description
eGrams_PerLiter	Grams per liter
eMilligrams_PerLiter	Milligrams per liter
eMicrograms_PerLiter	Micrograms per liter
eGrams_PerCubicMeter	Grams per cubic meter
eMilligrams_PerCubicMeter	Milligrams per cubic meter
eMicrograms_PerCubicMeter	Micrograms per cubic meter
eNanograms_PerCubicMeter	Nanograms per cubic meter
eGrams_PerCubicCentimeter	Grams per cubic centimeter
eBecquerels	Becquerels
eKilobecquerels	Kilobecquerels
eMegabecquerels	Megabecquerels
eGray	Grays (absorbed dose)
eMilligray	Milligrays (absorbed dose)
eMicrogray	Micrograys (absorbed dose)
eSieverts	Sieverts
eMillisieverts	Millisieverts
eMicrosieverts	Microsieverts
eMicrosievertsPerHour	Microsieverts per hour
eDecibels_A	Decibels A
eNephelometric_TurbidityUnit	Nephelometric turbidity unit NTU
ePH	pH
eGrams_PerSquareMeter	Grams per square meter
eMinutes_PerDegreeKelvin	Minutes per degree Kelvin

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7 Universal

4.2.1.7.1 E_BA_Action

Selection of the control direction of a controller.

Syntax

```
TYPE E_BA_Action:
(
  Invalid        := -1,
  eDirect        := 0,
  eReverse       := 1
)INT;
End_TYPE
```

Name	Description
Invalid	No significance for the user
eDirect	Direct control direction (cooling)
eReverse	Indirect control direction (heating)

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7.2 E_BA_Language

The enumeration is used to specify languages.

Syntax

```
TYPE E_BA_Language:
(
  Invalid     := 0,

  eEnglish    := 1,
  eGerman     := 2,
);
END_TYPE
```

Name	Description
Invalid	No significance for the user
eEnglish	English
eGerman	German

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7.3 E_BA_MeasuringElement

The enumeration is used to specify temperature measuring elements.

Syntax

```
TYPE E_BA_MeasuringElement :
(
  Undefined            := 0,

  eNI100               := 1,
  eNI120               := 2,
  eNI1000              := 3,
  eNI1000_LS           := 4,
  eNTC1K8              := 5,
  eNTC1K8_TK           := 6,
  eNTC2K2              := 7,
  eNTC3K               := 8,
  eNTC5K               := 9,
  eNTC10K              := 10,
  eNTC10KPRE           := 11,
  eNTC10K_3204         := 12,
  eNTC10KTYP2          := 13,
  eNTC10KTYP3          := 14,
  eNTC10KDALE          := 15,
  eNTC10K3A221         := 16,
  eNTC20K              := 17,
  eNTC100K             := 18,
  ePoti_Resolution_01  := 19,
  ePoti_Resolution_1   := 20,
  eOutput_10_5000      := 21,
  eOutput_10_1200      := 22,
  ePT100               := 23,
  ePT200               := 24,
  ePT500               := 25,
  ePT1000              := 26,
);
END_TYPE
```

Name	Description
Undefined	Not defined
eNI100	NI100
eNI120	NI120
eNI1000	NI1000
eNI1000_LS	RSNI1000 (NI1000 according to Landis&Staefa characteristic curve: 1000 Ω at 0 °C and 1500 Ω at 100 °C)
eNTC1K8	NTC1K8
eNTC1K8_TK	NTC1K8_TK
eNTC2K2	NTC2K2
eNTC3K	NTC3K
eNTC5K	NTC5K
eNTC10K	NTC10K
eNTC10KPRE	NTC10KPRE
eNTC10K_3204	NTC10K_3204
eNTC10KTYP2	NTC10KTYP2
eNTC10KTYP3	NTC10KTYP3
eNTC10KDALE	NTC10KDALE
eNTC10K3A221	NTC10K3A221
eNTC20K	NTC20K
eNTC100K	NTC100K
ePoti_Resolution_01	Potentiometer, resolution 0.1 Ω
ePoti_Resolution_1	Potentiometer, resolution 1 Ω
E_Output_10_5000	Output 10.0 Ω – 5000.0 Ω
E_Output_10_1200	Output 10.0 Ω – 1200.0 Ω
ePT100	PT100
ePT200	PT200
ePT500	PT500
ePT1000	PT1000

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7.4 E_BA_Polarity

The enumeration is used to specify the polarity (e.g. normally closed / normally open contact).

Syntax

```
TYPE E_BA_Polarity:
(
  Invalid        := -1,

  eNormal        := 0,
  eReverse       := 1
);
End_TYPE
```

Name	Description
Invalid	No significance for the user
eNormal	Normally open contact
eReverse	Normally closed contact

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7.5 E_BA_Reliability

The enumeration gives conclusions about the reliability of sensors or measurement data.

Syntax

```
TYPE E_BA_Reliability:
(
  Invalid                := 0,

  eNoFaultDetected       := 1,
  eNoSensor              := 1,
  eOverRange             := 2,
  eUnderRange            := 3,
  eOpenLoop              := 4,
  eShortedLoop           := 5,
  eNoOutput              := 6,
  eUnreliableOther       := 7,
  eProcessError          := 8,
  eMultiStateFault       := 9,
  eConfigurationError    := 10,
  eCommunicationFailure  := 12,
  eMemberFault           := 13
);
END_TYPE
```

Name	Description
Invalid	No significance for the user.
eNoFaultDetected	NO_FAULT_DETECTED No error described in this enumeration was detected.
eNoSensor	NO_SENSOR There is no sensor connected to the input object.
eOverRange	OVER_RANGE The measured value is above the normal measuring range.
eUnderRange	UNDER_RANGE The measured value is below the normal measuring range.
eOpenLoop	OPEN_LOOP A value is detected that indicates a wire break.
eShortedLoop	SHORTED_LOOP A value is detected that indicates a short circuit.
eNoOutput	NO_OUTPUT There is no output device connected to the output object.
eUnreliableOther	Internal: other plausibility error.
eProcessError	PROCESS_ERROR A process error has occurred.
eMultiStateFault	MULTI_STATE_FAULT The FAULT_STATE, FAULT_LIFE_SAFETY or FAULT_CHARACTERSTRING error algorithm has detected an error state.
eConfigurationError	CONFIGURATION_ERROR The properties of the object are not in a consistent state.
eCommunicationFailure	COMMUNICATION_FAILURE Communication error
eMemberFault	Fault member

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.1.7.6 E_BA_ToggleMode

The enumeration is used to interpret the function of a boolean change.

Syntax

```
TYPE E_BA_ToggleMode:
(
  Invalid        := 0,

  eSwitch        := 1,
  ePushButton    := 2
)BYTE;
END_TYPE
```

Name	Description
Invalid	Invalid, has no meaning.
eSwitch	The output value retains its value when changed. The object has assumed the function of a switch.
ePushButton	The output value changes its value for one cycle and then automatically restores the old state. The object has assumed the function of a push button.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2 Types

4.2.2.1 Calendar

4.2.2.1.1 ST_BA_CalendarEntry

Structure for specifying a calendar entry.

Syntax

```
TYPE ST_BA_CalendarEntry :
STRUCT
  eType             : E_BA_DateValChoice;
  uDate             : U_BA_DateVal;
END_STRUCT
END_TYPE
```

Name	Type	Description
eType	E_BA_DateValChoice [▶ 53]	Specification of the range selection
uDate	U_BA_DateVal [▶ 76]	Specification of the period

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2 Date and time

4.2.2.2.1 ST_BA_Date

Structure for the description of a date.

Syntax

```
TYPE ST_BA_Date :
STRUCT
  nYear          : BYTE          := 16#FF;
  eMonth         : E_BA_Month    := E_BA_Month.Unspecified;
  nDay           : E_BA_Day      := E_BA_Day.Unspecified;
  eDayOfWeek     : E_BA_Weekday := E_BA_Weekday.Unspecified;
END_STRUCT
END_TYPE
```

Name	Type	Description
nYear	BYTE	Year specification, counting from the year 1900.
eMonth	E_BA_Month [▶ 55]	Month specification
nDay	E_BA_Day [▶ 54]	Specification of the day in the month
eDayOfWeek	E_BA_Weekday [▶ 56]	Specification of the day of the week

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2.2 ST_BA_DateRange

Structure for describing a date range.

Syntax

```
TYPE ST_BA_DateRange :
STRUCT
  stDateFrom        : ST_BA_Date;
  stDateTo          : ST_BA_Date;
END_STRUCT
END_TYPE
```

Name	Type	Description
stDateFrom	ST_BA_Date [▶ 74]	Start of a period
stDateTo	ST_BA_Date [▶ 74]	End of a period

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2.3 ST_BA_DateTime

Structure for describing a date including a time.

Syntax

```
TYPE ST_BA_DateTime :
STRUCT
  stDate            : ST_BA_Date;
  stTime            : ST_BA_Time;
END_STRUCT
END_TYPE
```

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Specification of a date
stTime	ST_BA_Time [▶ 75]	Specification of the time of day

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2.4 ST_BA_Time

Structure to specify the time of day.

Syntax

```
TYPE ST_BA_Time :
STRUCT
  nHour               : BYTE := 16#FF;
  nMinute             : BYTE := 16#FF;
  nSecond             : BYTE := 16#FF;
  nHundredths         : BYTE := 16#FF;
END_STRUCT
END_TYPE
```

Name	Type	Description
nHour	BYTE	Specification of the hour
nMinute	BYTE	Specification of the minute
nSecond	BYTE	Specification of the second
nHundredths	BYTE	Specification of the hundredth of a second

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2.5 ST_BA_WeekNDay

Structure for specifying a day of the week.

Syntax

```
TYPE ST_BA_WeekNDay :
STRUCT
  eMonth          : E_BA_Month      := E_BA_Month.Unspecified;
  eWeekOfMonth    : E_BA_Week       := E_BA_Week.Unspecified;
  eWeekday        : E_BA_Weekday    := E_BA_Weekday.Unspecified;
END_STRUCT
END_TYPE
```

Name	Type	Description
eMonth	E_BA_Month [▶ 55]	Specification of the month
eWeekOfMonth	E_BA_Week [▶ 55]	Specification of the week within a month
eWeekday	E_BA_Weekday [▶ 56]	Specification of the day within a week

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.2.6 U_BA_DateVal

The data type UNION represents a date value, which can have different forms (*Date*, *Range*, *WeekNDay*).

The type of the value is specified by choice enum (E_BA_DateValChoice [▶ 53]) in the context (e.g. ST_BA_CalendarEntry [▶ 73] or ST_BA_ClassValue [▶ 80]).

All other elements start at the same address in the memory area and are also written to.
Their content, however, is then usually not meaningful.

Syntax

```
TYPE U_BA_DateVal :
UNION
  stDate          : ST_BA_Date;
  stDateRange     : ST_BA_DateRange;
  stWeekDay       : ST_BA_WeekDay;
END_UNION
END_TYPE
```

Name	Type	Description
stDate	ST_BA_Date [▶ 74]	Input as date
stDateRange	ST_BA_DateRange [▶ 74]	Input as date range
stWeekDay	ST_BA_WeekNDay [▶ 75]	Input as weekday

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.3 Event

4.2.2.3.1 ST_BA_EventTransitions

Structure for specifying a change of state of events.

Syntax

```
TYPE ST_BA_EventTransitions :
STRUCT
  bToOffNormal    : BOOL;
  bToFault        : BOOL;
  bToNormal       : BOOL;
END_STRUCT
END_TYPE
```

Name	Type	Description
bToOffNormal	BOOL	Change to the alarm state.
bToFault	BOOL	Change to the "invalid value" state.
bToNormal	BOOL	Change to the normal state.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.4 Schedule

4.2.2.4.1 ST_BA_SchedEntry

Structure for specifying a schedule entry.

Syntax

```
TYPE ST_BA_SchedEntry :
STRUCT
  eState          : E_BA_SchedEntryState;
  stTime          : ST_BA_Time;
  uValue          : U_BA_ClassValue;
END_STRUCT
END_TYPE
```

Name	Type	Description
eState	E_BA_SchedEntryState [▶ 58]	Specification of the input status
stTime	ST_BA_Time [▶ 75]	Specification of the time of day
uValue	U_BA_ClassValue [▶ 80]	Specification of the switch value

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.5 ST_BA_Byte

Structure for representing bits in a byte.

Syntax

```
TYPE ST_BA_Byte :
STRUCT
  bBit1    : BIT;
  bBit2    : BIT;
  bBit3    : BIT;
  bBit4    : BIT;
  bBit5    : BIT;
  bBit6    : BIT;
  bBit7    : BIT;
  bBit8    : BIT;
END_STRUCT
END_TYPE
```

Name	Type	Description
bBit1…bBit8	BIT	Display of the individual bits.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.6 ST_BA_EnumInfo

To describe enumerations in more detail textually, you can assign an element of the structure *ST_BA_EnumInfo* to each value of an enumeration in a global list. This is done, for example, in the global variable list BAComn_EnumDE [▶ 82].

The value of the enumeration refers to the element of the list that describes the element of the enumeration in more detail.

Syntax

```
TYPE ST_BA_EnumInfo :
STRUCT
  sName          : STRING;
  sDescription   : T_MaxString;
  sShortcut      : STRING(16);
END_STRUCT
END_TYPE
```

Name	Type	Description
sName	STRING	Name
sDescription	T_MaxString	Description
sShortcut	STRING(16)	Abbreviation

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.7 ST_BA_StatusFlags

Structure about the possible operating statuses of an object.

Syntax

```
TYPE ST_BA_StatusFlags :
STRUCT
  bInAlarm         : BOOL;
  bFault           : BOOL;
  bOverridden      : BOOL;
  bOutOfService    : BOOL;
END_STRUCT
END_TYPE
```

Name	Type	Description
bInAlarm	BOOL	Indicates an alarm state (offnormal).
bFault	BOOL	Indicates a reliability problem (fault).
bOverridden	BOOL	Indicates manual overwriting.
bOutOfService	BOOL	Indicates the out-of-service mode.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.8 ST_BA_Version

Structure for specifying the version number, see Revision control.

Syntax

```
TYPE ST_BA_Version : ARRAY [1 .. 4] OF UDINT;
END_TYPE
```

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.9 Trend

4.2.2.9.1 ST_BA_TrendEntry

Structure for describing a trend entry.

Syntax

```
TYPE ST_BA_TrendEntry :
STRUCT
  dtTime    : ST_BA_DateTime;
  eType     : E_BA_TrendEntryType     := E_BA_TrendEntryType.Invalid;
  stState   : ST_BA_StatusFlags;
  uValue    : U_BA_TrendEntryValue;
END_STRUCT
END_TYPE
```

Name	Type	Description
dtTime	ST_BA_DateTime [▶ 74]	Time specification
eType	E_BA_TrendEntryType [▶ 59]	Specification of the trend recording type: Binary, analog or multistate values.
stState	ST_BA_StatusFlags [▶ 78]	Possible statuses of the trend: alarm, faulty value, overridden, disconnected.
uValue	U_BA_TrendEntryValue [▶ 79]	Specification of the entry value including the status of the trend recording: start, stop, cleaned, interrupted.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.9.2 ST_BA_TrendEntryEvent

Structure for describing the status of the trend recording.

Syntax

```
TYPE ST_BA_TrendEntryEvent :
STRUCT
  bStart            : BIT;
  bStop             : BIT;
  bBufferPurged     : BIT;
  bInterrupted      : BIT;
END_STRUCT
END_TYPE
```

Name	Type	Description
bStart	BIT	Started
bStop	BIT	Stopped
bBufferPurged	BIT	Recording cleaned
bInterrupted	BIT	Recording interrupted

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.9.3 U_BA_TrendEntryValue

Structure for the value specification of trend entries.

The data type UNION allows the meaningful description of one of the containing elements.

All other elements start at the same address in the memory area and are also written to.
Their content, however, is then usually not meaningful.

Syntax

```
TYPE U_BA_TrendEntryValue EXTENDS U_BA_ClassValue:
UNION
  stEvent   : ST_BA_TrendEntryEvent;
END_UNION
END_TYPE
```

Name	Type	Description
stEvent	ST_BA_TrendEntryEvent [▶ 79]	Specification of the status of the trend recording.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.2.2.9.4 ST_BA_ClassValue

Structure for specifying an object value.

```
TYPE ST_BA_ClassValue :
STRUCT
  uValue   : U_BA_ClassValue;
  eClass   : E_BA_DataClass;
END_STRUCT
END_TYPE
```

Name	Type	Description
uValue	U_BA_ClassValue [▶ 56]	Specification of the value.
eClass	E_BA_DataClass [▶ 56]	Specification of the value type

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

U_BA_ClassValue [▶ 80]

4.2.2.9.5 U_BA_ClassValue

Structure for the input of values.

The data type UNION allows the meaningful description of one of the containing elements.

All other elements start at the same address in the memory area and are thus also written to.

Their content, however, is then usually not meaningful.

Syntax

```
TYPE U_BA_ClassValue :
UNION
  bVal      : BOOL;
  rVal      : REAL;
  udiVal    : UDINT;
END_UNION
END_TYPE
```

Name	Type	Description
bVal	BOOL	Binary values.
rVal	REAL	Analog floating point values.
udiVal	UDINT	Multistate value as UDINT.

Requirements

Development environment	Required PLC library
TwinCAT 3.1 4024.35	Tc2_BA2_Common from V2.1.20.0

4.3 GVLs

4.3.1 BAComn_Global

```
{attribute 'global_init_slot' := '49800'}
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
// Datatype Ranges:
 {warning disable C0196}
     nMinByte                : BYTE    := 16#00;
     nMaxByte                : BYTE    := 16#FF;
     nMinInt                 : INT     := 16#8000;
     nMaxInt                 : INT     := 16#7FFF;
     nMinUInt                : UINT    := 16#0000;
     nMaxUInt                : UINT    := 16#FFFF;
     nMinDInt                : DINT    := 16#80000000;
     nMaxDInt                : DINT    := 16#7FFFFFFF;
     nMinUDInt               : UDINT   := 16#00000000;
     nMaxUDInt               : UDINT   := 16#FFFFFFFF;
     fMinReal                : REAL    := -3.402823E+38;
     fMaxReal                : REAL    := 3.402823E+38;
     tMinTime                : TIME    := TO_TIME(0);
     tMaxTime                : TIME    := TO_TIME(16#FFFFFFFF);
     tMinTOD                 : TOD     := TO_TOD(0);
     tMaxTOD                 : TOD     := TO_TOD(16#FFFFFFFF);
     tMinDATE                : DATE    := TO_DATE(0);
     tMaxDATE                : DATE    := TO_DATE(16#FFFFFFFF);
     tMinDT                  : DT      := TO_DT(0);
     tMaxDT                  : DT      := TO_DT(16#FFFFFFFF);
 {warning restore C0196}

// I/O:
     nIO_RawMin              : INT      := 0;
     nIO_RawMax              : INT      := nMaxInt;
     nIO_Raw0V               : INT      := 0;                  // Raw value for 0V
     nIO_Raw1V               : INT      := (nIO_RawMax / 10); // Raw value for 1V
     nIO_Raw2V               : INT      := (nIO_Raw1V * 2);    // Raw value for 2V
     nIO_Raw3V               : INT      := (nIO_Raw1V * 3);    // Raw value for 3V
     nIO_Raw5V               : INT      := (nIO_Raw1V * 5);    // Raw value for 5V
     nIO_Raw10V              : INT      := (nIO_Raw1V * 10);   // Raw value for 10V
END_VAR

// General:
VAR_GLOBAL CONSTANT
 {region 'Time'}
     nMilli2Sek              : UINT    := 1000;
     nSek2Min                : UINT    := 60;
     nMin2Hour               : UINT    := 60;

     n24Hour2Hour            : UDINT   := (24 * 60 * 60);
     n24Hour2Milli           : UDINT   := n24Hour2Hour * 1000;

     udiMaxSecInMilli        : UDINT   := (nMaxUDInt / nMilli2Sek);     // Max. capable value (in [s]
) in a UDINT
     udiMaxMinInMilli        : UDINT   := (udiMaxSecInMilli / nSek2Min);// Max. capable value (in [m]
) in a UDINT
 {endregion}
 {region 'Characters'}
     bChar_0                 : BYTE    := 16#30;
     bChar_1                 : BYTE    := 16#31;
     bChar_2                 : BYTE    := 16#32;
     bChar_3                 : BYTE    := 16#33;
```

```
    bChar_4                 : BYTE    := 16#34;
    bChar_5                 : BYTE    := 16#35;
    bChar_6                 : BYTE    := 16#36;
    bChar_7                 : BYTE    := 16#37;
    bChar_8                 : BYTE    := 16#38;
    bChar_9                 : BYTE    := 16#39;
    bChar_Plus              : BYTE    := 16#2B;
    bChar_Minus             : BYTE    := 16#2D;
    bChar_Dot               : BYTE    := 16#2E;
 {endregion}
 {region 'Type'}
    fCloseToZero            : REAL    := 0.00001;   // Comparison value to prevent a division by zero
 {endregion}
 {region 'ADS'}
    tAmsNetID_Loopback      : T_AmsNetIdArr   := [ 127,0,0,1,1,1 ];
    sSymbolSeparator        : STRING(1)       := '.';
 {endregion}
END_VAR
```

4.3.2 BAComn_Param

```
{attribute 'qualified_only'}
VAR_GLOBAL CONSTANT
  {region 'Tokenizer'}
    nStrTokenizer_BufferSize  : UINT    := 32;
    nStrTokenizer_MaxLevel    : BYTE    := 5;
  {endregion}
END_VAR
```

Name	Type	Description
nStrTokenizer_BufferSize	UINT	Number of entries.
nStrTokenizer_MaxLevel	BYTE	Maximum depth of the token hierarchy.

4.3.3 Enumerations

4.3.3.1 BAComn_EnumDE

```
{attribute 'qualified_only'}
VAR_GLOBAL

aUnits    : ARRAY[E_BA_Unit.First .. E_BA_Unit.Last] OF ST_BA_EnumInfo := [

  (*  eArea_SquareMeters                  *) (sName := 'Square Meters',                       sDesc
ription := 'Fläche',                                    sShortcut := 'm²'),
  (*  eArea_SquareFeet                    *) (sName := 'Square Feet',                         sDesc
ription := '',                                          sShortcut := ''),
  (*  eElectrical_Milliamperes            *) (sName := 'Milliamperes',                        sDesc
ription := 'Strom',                                     sShortcut := 'mA'),
  (*  eElectrical_Amperes                 *) (sName := 'Amperes',                             sDesc
ription := 'Strom',                                     sShortcut := 'A'),
  (*  eElectrical_Ohms                    *) (sName := 'Ohms',                                sDesc
ription := 'Elektrischer Widerstand',                   sShortcut := 'O'),
  (*  eElectrical_Volts                   *) (sName := 'Volts',                               sDesc
ription := 'Elektrische Spannung',                      sShortcut := 'V'),
  (*  eElectrical_Kilovolts               *) (sName := 'Kilovolts',                           sDesc
ription := 'Elektrische Spannung',                      sShortcut := 'kV'),
  (*  eElectrical_Megavolts               *) (sName := 'Megavolts',                           sDesc
ription := 'Elektrische Spannung',                      sShortcut := 'MV'),
  (*  eElectrical_VoltAmperes             *) (sName := 'Volt Amperes',                        sDesc
ription := 'Elektrische Scheinleistung',                sShortcut := 'VA'),
  (*  eElectrical_KilovoltAmperes         *) (sName := 'Kilovolt Amperes',                    sDesc
ription := 'Elektrische Scheinleistung',                sShortcut := 'kVA'),
  (*  eElectrical_MegavoltAmperes         *) (sName := 'Megavolt Amperes',                    sDesc
ription := 'Elektrische Scheinleistung',                sShortcut := 'MVA'),
  (*  eElectrical_VoltAmperesReactive     *) (sName := 'Volt Amperes Reactive',               sDesc
ription := 'Elektrische Blindleistung',                 sShortcut := 'var'),
  (*  eElectrical_KilovoltAmperesReactive *) (sName := 'Kilovolt Amperes Reactive',           sDesc
ription := 'Elektrische Blindleistung',                 sShortcut := 'kvar'),
  (*  eElectrical_MegavoltAmperesReactive *) (sName := 'Megavolt Amperes Reactive',           sDesc
ription := 'Elektrische Blindleistung',                 sShortcut := 'Mvar'),
  (*  eElectrical_DegreesPhase            *) (sName := 'Degrees Phase',                       sDesc
```

```
ription := '',                                                  sShortcut := ''),
  (*  eElectrical_PowerFactor              *) (sName := 'Power Factor',                          sDesc
ription := 'phi Leistungsfaktor',                               sShortcut := 'cos'),
  (*  eEnergy_Joules                      *) (sName := 'Joules',                                sDesc
ription := 'Energie',                                           sShortcut := 'J'),
  (*  eEnergy_Kilojoules                  *) (sName := 'Kilojoules',                            sDesc
ription := 'Energie',                                           sShortcut := 'kJ'),
  (*  eEnergy_WattHours                   *) (sName := 'Watt Hours',                            sDesc
ription := 'Energie',                                           sShortcut := 'Wh'),
  (*  eEnergy_KilowattHours               *) (sName := 'Kilowatt Hours',                        sDesc
ription := 'Energie',                                           sShortcut := 'kWh'),
  (*  eEnergy_Btus                        *) (sName := 'Btus',                                  sDesc
ription := '',                                                  sShortcut := 'btus'),
  (*  eEnergy_Therms                      *) (sName := 'Therms',                                sDesc
ription := '',                                                  sShortcut := ''),
  (*  eEnergy_TonHours                    *) (sName := 'Ton Hours',                             sDesc
ription := '',                                                  sShortcut := ''),
  (*  eEnthalpy_JoulesPerKilogramDryAir   *) (sName := 'Joules per Kilogram Dry Air',           sDesc
ription := 'Enthalpie',                                         sShortcut := 'J/kg'),
  (*  eEnthalpy_BtusPerPoundDryAir        *) (sName := 'Btus per Pound Dry Air',                sDesc
ription := '',                                                  sShortcut := ''),
  (*  eFrequency_CyclesPerHour            *) (sName := 'Cycles per Hour',                       sDesc
ription := 'Schalthäufigkeit (Zyklen pro Stunde)',              sShortcut := '1/h'),
  (*  eFrequency_CyclesPerMinute          *) (sName := 'Cycles per Minute',                     sDesc
ription := 'Schalthäufigkeit (Zyklen pro Minute)',              sShortcut := '1/min'),
  (*  eFrequency_Hertz                    *) (sName := 'Hertz',                                 sDesc
ription := 'Frequenz',                                          sShortcut := 'Hz'),
  (*  eHumidity_GramsOfWaterPerKilogramDryAir*) (sName := 'Grams of Water per Kilogram Dry Air', sDe
scription := 'Wassergehalt',                                      sShortcut := 'g/kg tr. Luft'),
  (*  eHumidity_PercentRelativeHumidity   *) (sName := 'Percent Relative Humidity',             sDesc
ription := 'Relative Luftfeuchtigkeit',                         sShortcut := '% r. F.'),
  (*  eLength_Millimeters                  *) (sName := 'Millimeters',                           sDesc
ription := 'Länge',                                             sShortcut := 'mm'),
  (*  eLength_Meters                      *) (sName := 'Meters',                                sDesc
ription := 'Länge',                                             sShortcut := 'm'),
  (*  eLength_Inches                      *) (sName := 'Inches',                                sDesc
ription := 'Zoll',                                              sShortcut := ''),
  (*  eLength_Feet                        *) (sName := 'Feet',                                  sDesc
ription := 'Fuss',                                              sShortcut := ''),
  (*  eLight_WattsPerSquareFoot           *) (sName := 'Watts per Square Foot',                 sDesc
ription := '',                                                  sShortcut := ''),
  (*  eLight_WattsPerSquareMeter          *) (sName := 'Watts per Square Meter',                sDesc
ription := 'Spezifische Leistung',                              sShortcut := 'W/m²'),
  (*  eLight_Lumens                       *) (sName := 'Lumens',                                sDesc
ription := 'Lichtstrom',                                        sShortcut := 'lm'),
  (*  eLight_Luxes                        *) (sName := 'Luxes',                                 sDesc
ription := 'Beleuchtungsstärke',                                sShortcut := 'lx'),
  (*  eLight_FootCandles                  *) (sName := 'Foot Candles',                          sDesc
ription := '',                                                  sShortcut := ''),
  (*  eMass_Kilograms                     *) (sName := 'Kilograms',                             sDesc
ription := 'Masse',                                             sShortcut := 'kg'),
  (*  eMass_PoundsMass                    *) (sName := 'Pounds Mass',                           sDesc
ription := '',                                                  sShortcut := ''),
  (*  eMass_Tons                          *) (sName := 'Tons',                                  sDesc
ription := 'Masse',                                             sShortcut := 't'),
  (*  eMassFlow_KilogramsPerSecond        *) (sName := 'Kilograms per Second',                  sDesc
ription := 'Massenstrom',                                       sShortcut := 'kg/s'),
  (*  eMassFlow_KilogramsPerMinute        *) (sName := 'Kilograms per Minute',                  sDesc
ription := 'Massenstrom',                                       sShortcut := 'kg/min'),
  (*  eMassFlow_KilogramsPerHour          *) (sName := 'Kilograms per Hour',                    sDesc
ription := 'Massenstrom',                                       sShortcut := 'kg/h'),
  (*  eMassFlow_PoundsMassPerMinute       *) (sName := 'Pounds Mass per Minute',                sDesc
ription := '',                                                  sShortcut := ''),
  (*  eMassFlow_PoundsMassPerHour         *) (sName := 'Pounds Mass per Hour',                  sDesc
ription := '',                                                  sShortcut := ''),
  (*  ePower_Watts                        *) (sName := 'Watts',                                 sDesc
ription := 'Leistung',                                          sShortcut := 'W'),
  (*  ePower_Kilowatts                    *) (sName := 'Kilowatts',                             sDesc
ription := 'Leistung',                                          sShortcut := 'kW'),
  (*  ePower_Megawatts                    *) (sName := 'Megawatts',                             sDesc
ription := 'Leistung',                                          sShortcut := 'MW'),
  (*  ePower_BtusPerHour                  *) (sName := 'Btus per Hour',                         sDesc
ription := '',                                                  sShortcut := ''),
  (*  ePower_Horsepower                   *) (sName := 'Horsepower',                            sDesc
ription := 'Leistung',                                          sShortcut := 'PS'),
  (*  ePower_TonsRefrigeration            *) (sName := 'Tons Refrigeration',                    sDesc
ription := '',                                                  sShortcut := ''),
  (*  ePressure_Pascals                   *) (sName := 'Pascals',                               sDesc
ription := 'Druck',                                             sShortcut := 'Pa'),
  (*  ePressure_Kilopascals               *) (sName := 'Kilopascals',                           sDesc
```

```
ription := 'Druck',                                                    sShortcut := 'kPa'),
  (*  ePressure_Bars                        *) (sName := 'Bars',                                    sDesc
ription := 'Druck',                                                    sShortcut := 'bar'),
  (*  ePressure_PoundsForcePerSquareInch    *) (sName := 'Pounds Force per Square Inch',            sDesc
ription := '',                                                         sShortcut := ''),
  (*  ePressure_CentimetersOfWater          *) (sName := 'Centimeters of Water',                    sDesc
ription := 'Druck',                                                    sShortcut := ''),
  (*  ePressure_InchesOfWater               *) (sName := 'Inches of Water',                         sDesc
ription := '',                                                         sShortcut := ''),
  (*  ePressure_MillimetersOfMercury        *) (sName := 'Millimeters of Mercury',                  sDesc
ription := 'Hg Druck',                                                 sShortcut := 'mm'),
  (*  ePressure_CentimetersOfMercury        *) (sName := 'Centimeters of Mercury',                  sDesc
ription := 'Druck',                                                    sShortcut := ''),
  (*  ePressure_InchesOfMercury             *) (sName := 'Inches of Mercury',                       sDesc
ription := '',                                                         sShortcut := ''),
  (*  eTemperature_DegreesCelsius           *) (sName := 'Degrees Celsius',                         sDesc
ription := 'Temperatur',                                               sShortcut := '°C'),
  (*  eTemperature_DegreesKelvin            *) (sName := 'Degrees Kelvin',                          sDesc
ription := 'Temperatur (thermodynamisch)',                             sShortcut := 'K'),
  (*  eTemperature_DegreesFahrenheit        *) (sName := 'Degrees Fahrenheit',                      sDesc
ription := '',                                                         sShortcut := ''),
  (*  eTemperature_DegreeDaysCelsius        *) (sName := 'Degree Days Celsius',                     sDesc
ription := 'Gradtage',                                                 sShortcut := 'K d/c'),
  (*  eTemperature_DegreeDaysFahrenheit     *) (sName := 'Degree Days Fahrenheit',                  sDesc
ription := '',                                                         sShortcut := ''),
  (*  eTime_Years                           *) (sName := 'Years',                                   sDesc
ription := 'Zeit',                                                     sShortcut := 'a'),
  (*  eTime_Months                          *) (sName := 'Months',                                  sDesc
ription := 'Zeit',                                                     sShortcut := 'Monat'),
  (*  eTime_Weeks                           *) (sName := 'Weeks',                                   sDesc
ription := 'Zeit',                                                     sShortcut := 'Woche'),
  (*  eTime_Days                            *) (sName := 'Days',                                    sDesc
ription := 'Zeit',                                                     sShortcut := 'd'),
  (*  eTime_Hours                           *) (sName := 'Hours',                                   sDesc
ription := 'Zeit',                                                     sShortcut := 'h'),
  (*  eTime_Minutes                         *) (sName := 'Minutes',                                 sDesc
ription := 'Zeit',                                                     sShortcut := 'min'),
  (*  eTime_Seconds                         *) (sName := 'Seconds',                                 sDesc
ription := 'Zeit',                                                     sShortcut := 's'),
  (*  eVelocity_MetersPerSecond             *) (sName := 'Meters per Second',                       sDesc
ription := 'Geschwindigkeit',                                          sShortcut := 'm/s'),
  (*  eVelocity_KilometersPerHour           *) (sName := 'Kilometers per Hour',                     sDesc
ription := 'Geschwindigkeit',                                          sShortcut := 'km/h'),
  (*  eVelocity_FeetPerSecond               *) (sName := 'Feet per Second',                         sDesc
ription := '',                                                         sShortcut := ''),
  (*  eVelocity_FeetPerMinute               *) (sName := 'Feet per Minute',                         sDesc
ription := '',                                                         sShortcut := ''),
  (*  eVelocity_MilesPerHour                *) (sName := 'Miles per Hour',                          sDesc
ription := '',                                                         sShortcut := ''),
  (*  eVolume_CubicFeet                     *) (sName := 'Cubic Feet',                              sDesc
ription := '',                                                         sShortcut := ''),
  (*  eVolume_CubicMeters                   *) (sName := 'Cubic Meters',                            sDesc
ription := 'Volumen',                                                  sShortcut := 'm³'),
  (*  eVolume_ImperialGallons               *) (sName := 'Imperial Gallons',                        sDesc
ription := 'Brit. Gallonen',                                           sShortcut := ''),
  (*  eVolume_Liters                        *) (sName := 'Liters',                                  sDesc
ription := 'Volumen',                                                  sShortcut := 'l'),
  (*  eVolume_UsGallons                     *) (sName := 'US Gallons',                              sDesc
ription := 'US-Gallonen',                                              sShortcut := ''),
  (*  eVolumetricFlow_CubicFeetPerMinute    *) (sName := 'Cubic Feet per Minute',                   sDesc
ription := 'Volumenstrom',                                             sShortcut := 'cf/min'),
  (*  eVolumetricFlow_CubicMetersPerSecond *) (sName := 'Cubic Meters per Second',                  sDesc
ription := 'Volumenstrom',                                             sShortcut := 'm³/s'),
  (*  eVolumetricFlow_ImperialGallonsPerMinute*) (sName := 'Imperial Gallons per Minute',           sDesc
ription := '',                                                         sShortcut := ''),
  (*  eVolumetricFlow_LitersPerSecond       *) (sName := 'Liters per Second',                       sDesc
ription := 'Volumenstrom',                                             sShortcut := 'l/s'),
  (*  eVolumetricFlow_LitersPerMinute       *) (sName := 'Liters per Minute',                       sDesc
ription := 'Volumenstrom',                                             sShortcut := 'l/min'),
  (*  eVolumetricFlow_UsGallonsPerMinute    *) (sName := 'US Gallons per Minute',                   sDesc
ription := '',                                                         sShortcut := ''),
  (*  eOther_DegreesAngular                 *) (sName := 'Degrees Angular',                         sDesc
ription := 'Winkelgrade',                                              sShortcut := '°'),
  (*  eOther_DegreesCelsiusPerHour          *) (sName := 'Degrees Celsius per Hour',                sDesc
ription := 'Temperaturänderung pro Zeiteinheit',                       sShortcut := '°C/h'),
  (*  eOther_DegreesCelsiusPerMinute        *) (sName := 'Degrees Celsius per Minute',              sDesc
ription := 'Temperaturänderung pro Zeiteinheit',                       sShortcut := '°C/min'),
  (*  eOther_DegreesFahrenheitPerHour       *) (sName := 'Degrees Fahrenheit per Hour',             sDesc
ription := '',                                                         sShortcut := ''),
  (*  eOther_DegreesFahrenheitPerMinute     *) (sName := 'Degrees Fahrenheit per Minute',           sDesc
```

```
ription := '',                                                  sShortcut := ''),
  (*  eOther_NoUnits                       *) (sName := 'No Units',                              sDesc
ription := '',                                                  sShortcut := ''),
  (*  eOther_PartsPerMillion              *) (sName := 'Parts per Million',                     sDesc
ription := 'Konzentration',                                     sShortcut := 'ppm'),
  (*  eOther_PartsPerBillion              *) (sName := 'Parts per Billion',                     sDesc
ription := 'Konzentration',                                     sShortcut := 'ppb'),
  (*  eOther_Percent                      *) (sName := 'Percent',                               sDesc
ription := 'Prozent',                                           sShortcut := '%'),
  (*  eOther_PercentPerSecond             *) (sName := 'Percent per Second',                    sDesc
ription := 'Änderungsgeschwindigkeit',                          sShortcut := '%/s'),
  (*  eOther_PerMinute                    *) (sName := 'Per Minute',                            sDesc
ription := 'Drehzahl',                                          sShortcut := '1/m'),
  (*  eOther_PerSecond                    *) (sName := 'Per Second',                            sDesc
ription := 'Drehzahl',                                          sShortcut := '1/s'),
  (*  eOther_PsiPerDegreeFahrenheit       *) (sName := 'Psi per Degree Fahrenheit',             sDesc
ription := '',                                                  sShortcut := ''),
  (*  eOther_Radians                      *) (sName := 'Radians',                               sDesc
ription := 'Bogenmaß',                                          sShortcut := 'rad'),
  (*  eOther_RevolutionsPerMinute         *) (sName := 'Revolutions per Minute',                sDesc
ription := 'Umdrehungen pro Minute',                            sShortcut := '1/min'),
  (*  eCurrency_Currency1                 *) (sName := 'Currency 1',                            sDesc
ription := 'Währung',                                           sShortcut := '€'),
  (*  eCurrency_Currency2                 *) (sName := 'Currency 2',                            sDesc
ription := 'Währung',                                           sShortcut := 'DM'),
  (*  eCurrency_Currency3                 *) (sName := 'Currency 3',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency4                 *) (sName := 'Currency 4',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency5                 *) (sName := 'Currency 5',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency6                 *) (sName := 'Currency 6',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency7                 *) (sName := 'Currency 7',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency8                 *) (sName := 'Currency 8',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency9                 *) (sName := 'Currency 9',                            sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eCurrency_Currency10                *) (sName := 'Currency 10',                           sDesc
ription := 'Währung',                                           sShortcut := ''),
  (*  eArea_SquareInches                  *) (sName := 'Square Inches',                         sDesc
ription := '',                                                  sShortcut := ''),
  (*  eArea_SquareCentimeters             *) (sName := 'Square Centimeters',                    sDesc
ription := 'Fläche',                                            sShortcut := 'cm²'),
  (*  eEnthalpy_BtusPerPound              *) (sName := 'Btus per Pound',                        sDesc
ription := '',                                                  sShortcut := ''),
  (*  eLength_Centimeters                 *) (sName := 'Centimeters',                           sDesc
ription := 'Länge',                                             sShortcut := 'cm'),
  (*  eMassFlow_PoundsMassPerSecond       *) (sName := 'Pounds Mass per Second',                sDesc
ription := '',                                                  sShortcut := ''),
  (*  eTemperature_DeltaDegreesFahrenheit *) (sName := 'Delta Degrees Fahrenheit',              sDesc
ription := '',                                                  sShortcut := ''),
  (*  eTemperature_DeltaDegreesKelvin     *) (sName := 'Delta Degrees Kelvin',                  sDesc
ription := 'Temperaturdifferenz',                               sShortcut := 'delta K'),
  (*  eElectrical_Kilohms                 *) (sName := 'Kilohms',                               sDesc
ription := 'Spezifischer elektrischer Widerstand',              sShortcut := 'kOhm'),
  (*  eElectrical_Megohms                 *) (sName := 'Megohms',                               sDesc
ription := 'Spezifischer elektrischer Widerstand',              sShortcut := 'MOhm'),
  (*  eElectrical_Millivolts              *) (sName := 'Millivolts',                            sDesc
ription := 'Elektrische Spannung',                              sShortcut := 'mV'),
  (*  eEnergy_KilojoulesPerKilogram       *) (sName := 'Kilojoules per Kilogram',               sDesc
ription := 'Enthalpie',                                         sShortcut := 'kJ/kg'),
  (*  eEnergy_Megajoules                  *) (sName := 'Megajoules',                            sDesc
ription := 'Energie',                                           sShortcut := 'MJ'),
  (*  eEntropy_JoulesPerDegreeKelvin      *) (sName := 'Joules per Degree Kelvin',              sDesc
ription := 'Entropie',                                          sShortcut := 'J/K'),
  (*  eEntropy_JoulesPerKilogramDegreeKelvin*) (sName := 'Joules per Kilogram Degree Kelvin',   sDes
cription := 'Spezifische Entropie',                              sShortcut := 'J/(kg K)'),
  (*  eFrequency_Kilohertz                *) (sName := 'Kilohertz',                             sDesc
ription := 'Frequenz',                                          sShortcut := 'kHz'),
  (*  eFrequency_Megahertz                *) (sName := 'Megahertz',                             sDesc
ription := 'Frequenz',                                          sShortcut := 'MHz'),
  (*  eFrequency_PerHour                  *) (sName := 'Per Hour',                              sDesc
ription := 'Drehzahl',                                          sShortcut := '1/h'),
  (*  ePower_Milliwatts                   *) (sName := 'Milliwatts',                            sDesc
ription := 'Leistung',                                          sShortcut := 'mW'),
  (*  ePressure_Hectopascals              *) (sName := 'Hectopascals',                          sDesc
ription := 'Druck',                                             sShortcut := 'hPa'),
  (*  ePressure_Millibars                 *) (sName := 'Millibars',                             sDesc
```

```
ription := 'Druck',                                                sShortcut := 'mbar'),
  (*  eVolumetricFlow_CubicMetersPerHour   *) (sName := 'Cubic Meters per Hour',                sDesc
ription := 'Volumenstrom',                                         sShortcut := 'm³/h'),
  (*  eVolumetricFlow_LitersPerHour         *) (sName := 'Liters per Hour',                     sDesc
ription := 'Volumenstrom',                                         sShortcut := 'l/h'),
  (*  eOther_KilowattHoursPerSquareMeter    *) (sName := 'Kilowatt Hours per Square Meter',     sDesc
ription := 'Energiebedarfskennwert',                               sShortcut := 'kWh/m²'),
  (*  eOther_KilowattHoursPerSquareFoot     *) (sName := 'Kilowatt Hours per Square Foot',      sDesc
ription := '',                                                     sShortcut := ''),
  (*  eOther_MegajoulesPerSquareMeter       *) (sName := 'Megajoules per Square Meter',         sDesc
ription := 'Energiebedarfskennwert',                               sShortcut := 'MJ/m²'),
  (*  eOther_MegajoulesPerSquareFoot        *) (sName := 'Megajoules per Square Foot',          sDesc
ription := '',                                                     sShortcut := ''),
  (*  eOther_WattsPerSquareMeterDegreeKelvin*) (sName := 'Watts per Square Meter Degree Kelvin',sDes
cription := 'Wärmedurchgangskoeffizient',                           sShortcut := 'W/(m² K)'),
  (*  eVolumetricFlow_CubicFeetPerSecond    *) (sName := 'Cubic Feet per Second',               sDesc
ription := 'Volumenstrom',                                         sShortcut := 'cf/sec'),
  (*  eOther_PercentObscurationPerFoot      *) (sName := 'Percent Obscuration per Foot',        sDesc
ription := '',                                                     sShortcut := ''),
  (*  eOther_PercentObscurationPerMeter     *) (sName := 'Percent Obscuration per Meter',       sDesc
ription := 'Verdunkelung (Rauchmelder)',                           sShortcut := '%/m'),
  (*  eElectrical_Milliohms                 *) (sName := 'Milliohms',                           sDesc
ription := 'Spezifischer elektrischer Widerstand',                 sShortcut := 'mOhm'),
  (*  eEnergy_MegawattHours                 *) (sName := 'Megawatt Hours',                      sDesc
ription := 'Energie',                                              sShortcut := 'MWh'),
  (*  eEnergy_KiloBtus                      *) (sName := 'Kilo Btus',                           sDesc
ription := '',                                                     sShortcut := ''),
  (*  eEnergy_MegaBtus                      *) (sName := 'Mega Btus',                           sDesc
ription := '',                                                     sShortcut := ''),
  (*  eEnthalpy_KilojoulesPerKilogramDryAir*) (sName := 'Kilojoules per Kilogram Dry Air',      sDesc
ription := 'Enthalpie',                                            sShortcut := 'kJ/kg tr. Luft'),
  (*  eEnthalpy_MegajoulesPerKilogramDryAir*) (sName := 'Megajoules per Kilogram Dry Air',      sDesc
ription := 'Enthalpie',                                            sShortcut := 'MJ/kg tr. Luft'),
  (*  eEntropy_KilojoulesPerDegreeKelvin    *) (sName := 'Kilojoules per Degree Kelvin',        sDesc
ription := 'Entropie',                                             sShortcut := 'kJ/K'),
  (*  eEntropy_MegajoulesPerDegreeKelvin    *) (sName := 'Megajoules per Degree Kelvin',        sDesc
ription := 'Entropie',                                             sShortcut := 'MJ/K'),
  (*  eForce_Newton                         *) (sName := 'Newton',                              sDesc
ription := 'Kraft',                                                sShortcut := 'N'),
  (*  eMassFlow_GramsPerSecond              *) (sName := 'Grams per Second',                    sDesc
ription := 'Massenstrom',                                          sShortcut := 'g/s'),
  (*  eMassFlow_GramsPerMinute              *) (sName := 'Grams per Minute',                    sDesc
ription := 'Massenstrom',                                          sShortcut := 'g/min'),
  (*  eMassFlow_TonsPerHour                 *) (sName := 'Tons per Hour',                       sDesc
ription := 'Massenstrom',                                          sShortcut := 't/h'),
  (*  ePower_KiloBtusPerHour                *) (sName := 'Kilo Btus per Hour',                  sDesc
ription := '',                                                     sShortcut := ''),
  (*  eTime_HundredthsSeconds               *) (sName := 'Hundredths Seconds',                  sDesc
ription := 'Zeit',                                                 sShortcut := '10-² s'),
  (*  eTime_Milliseconds                    *) (sName := 'Milliseconds',                        sDesc
ription := 'Zeit',                                                 sShortcut := 'ms'),
  (*  eTorque_NewtonMeters                  *) (sName := 'Torque Newton Meters',                sDesc
ription := 'Drehmoment',                                           sShortcut := 'Nm'),
  (*  eVelocity_MillimetersPerSecond        *) (sName := 'Millimeters per Second',              sDesc
ription := 'Geschwindigkeit',                                      sShortcut := 'mm/s'),
  (*  eVelocity_MillimetersPerMinute        *) (sName := 'Millimeters per Minute',              sDesc
ription := 'Geschwindigkeit',                                      sShortcut := 'mm/min'),
  (*  eVelocity_MetersPerMinute             *) (sName := 'Meters per Minute',                   sDesc
ription := 'Geschwindigkeit',                                      sShortcut := 'm/min'),
  (*  eVelocity_MetersPerHour               *) (sName := 'Meters per Hour',                     sDesc
ription := 'Geschwindigkeit',                                      sShortcut := 'm/h'),
  (*  eVolumetricFlow_CubicMetersPerMinute *) (sName := 'Cubic Meters per Minute',              sDesc
ription := 'Volumenstrom',                                         sShortcut := 'm³/min'),
  (*  eAcceleration_MetersPerSecondPerSecond*) (sName := 'Meters per Second per Second',        sDes
cription := 'Beschleunigung',                                       sShortcut := 'm/s²'),
  (*  eElectrical_AmperesPerMeter           *) (sName := 'Amperes per Meter',                  sDesc
ription := 'Strom pro Länge',                                      sShortcut := 'A/m'),
  (*  eElectrical_AmperesPerSquareMeter     *) (sName := 'Amperes per Square Meter',           sDesc
ription := 'Strom pro Fläche',                                     sShortcut := 'A/m²'),
  (*  eElectrical_AmpereSquareMeters        *) (sName := 'Ampere Square Meters',               sDesc
ription := 'Strom mal Fläche',                                     sShortcut := 'Am²'),
  (*  eElectrical_Farads                    *) (sName := 'Farads',                             sDesc
ription := 'Elektrische Kapazität',                                sShortcut := 'F'),
  (*  eElectrical_Henrys                    *) (sName := 'Henrys',                             sDesc
ription := 'Induktivität',                                         sShortcut := 'H'),
  (*  eElectrical_OhmMeters                 *) (sName := 'Ohm Meters',                         sDesc
ription := 'Spezifischer elektrischer Widerstand',                 sShortcut := 'Ohmm'),
  (*  eElectrical_Siemens                   *) (sName := 'Siemens',                            sDesc
ription := 'Elektrischer Leitwert',                                sShortcut := 'S'),
  (*  eElectrical_SiemensPerMeter           *) (sName := 'Siemens per Meter',                  sDesc
```

```
ription := 'Elektrische Leitfähigkeit',                          sShortcut := 'S/m'),
  (*  eElectrical_Teslas                    *) (sName := 'Teslas',                                sDesc
ription := 'Magnetische Flussdichte',                            sShortcut := 'T'),
  (*  eElectrical_VoltsPerDegreeKelvin      *) (sName := 'Volts per Degree Kelvin',               sDesc
ription := 'Thermoelektrische Spannung',                         sShortcut := 'V/K'),
  (*  eElectrical_VoltsPerMeter             *) (sName := 'Volts per Meter',                       sDesc
ription := 'Elektrische Feldstärke',                             sShortcut := 'V/m'),
  (*  eElectrical_Webers                     *) (sName := 'Webers',                                sDesc
ription := 'Magnetischer Fluss',                                 sShortcut := 'Wb'),
  (*  eLight_Candelas                        *) (sName := 'Candelas',                              sDesc
ription := 'Lichtstärke',                                        sShortcut := 'cd'),
  (*  eLight_CandelasPerSquareMeter         *) (sName := 'Candelas per Square Meter',             sDesc
ription := 'Leuchtdichte',                                       sShortcut := 'cd/m²'),
  (*  eTemperature_DegreesKelvinPerHour      *) (sName := 'Degrees Kelvin per Hour',               sDesc
ription := 'Temperaturänderung pro Zeit',                        sShortcut := 'K/h'),
  (*  eTemperature_DegreesKelvinPerMinute    *) (sName := 'Degrees Kelvin per Minute',             sDesc
ription := 'Temperaturänderung pro Zeit',                        sShortcut := 'K/min'),
  (*  eOther_JouleSeconds                    *) (sName := 'Joule Seconds',                         sDesc
ription := 'Drehimpuls',                                         sShortcut := 'Js'),
  (*  eOther_RadiansPerSecond                *) (sName := 'Radians per Second',                    sDesc
ription := 'Winkelgeschwindigkeit',                              sShortcut := 'rad/s'),
  (*  eOther_SquareMetersPerNewton           *) (sName := 'Square Meters per Newton',              sDesc
ription := 'Kraftverteilung',                                    sShortcut := 'm²/N'),
  (*  eOther_KilogramsPerCubicMeter          *) (sName := 'Kilograms per Cubic Meter',             sDesc
ription := 'Dichte',                                             sShortcut := 'kg/m³'),
  (*  eOther_NewtonSeconds                   *) (sName := 'Newton Seconds',                        sDesc
ription := 'Impuls',                                             sShortcut := 'Ns'),
  (*  eOther_NewtonsPerMeter                 *) (sName := 'Newtons per Meter',                     sDesc
ription := 'Oberflächenspannung',                                sShortcut := 'N/m'),
  (*  eOther_WattsPerMeterPerDegreeKelvin    *) (sName := 'Watts per Meter per Degree Kelvin',     sDesc
ription := 'Wärmeleitfähigkeit',                                 sShortcut := 'W/m K'),
  (*  eMicro_Siemens                         *) (sName := 'Micro Siemens',                         sDesc
ription := 'Elektrischer Leitwert',                              sShortcut := 'µS'),
  (*  eCubic_FeetPerHour                     *) (sName := 'Cubic Feet per Hour',                   sDesc
ription := 'Durchsatz',                                          sShortcut := 'cf/h'),
  (*  eUs_GallonsPerHour                     *) (sName := 'US Gallons per Hour',                   sDesc
ription := 'Durchsatz',                                          sShortcut := 'G/h'),
  (*  eKilometers                            *) (sName := 'Kilometers',                            sDesc
ription := 'Länge',                                              sShortcut := 'km'),
  (*  eMicrometers                           *) (sName := 'Micrometers',                           sDesc
ription := 'Länge',                                              sShortcut := 'µm'),
  (*  eGrams                                 *) (sName := 'Grams',                                 sDesc
ription := 'Masse',                                              sShortcut := 'g'),
  (*  eMilligrams                            *) (sName := 'Milligrams',                            sDesc
ription := 'Masse',                                              sShortcut := 'mg'),
  (*  eMilliliters                           *) (sName := 'Milliliters',                           sDesc
ription := 'Volumen',                                            sShortcut := 'ml'),
  (*  eMillilitersPerSecond                  *) (sName := 'Milliliters per Second',                sDesc
ription := 'Volumenstrom',                                       sShortcut := 'ml/s'),
  (*  eDecibels                              *) (sName := 'Decibels',                              sDesc
ription := 'Schalldruckpegel',                                   sShortcut := 'dB'),
  (*  eDecibelsMillivolt                     *) (sName := 'Decibels Millivolt',                    sDesc
ription := 'Spannungspegel (bez.auf 1V)',                        sShortcut := 'dBV'),
  (*  eDecibelsVolt                          *) (sName := 'Decibels Volt',                         sDesc
ription := 'Spannungspegel (bez.auf 1mV)',                       sShortcut := 'dBmV'),
  (*  eMillisiemens                          *) (sName := 'Millisiemens',                          sDesc
ription := 'Elektrischer Leitwert',                              sShortcut := 'mS'),
  (*  eWatt_HoursReactive                    *) (sName := 'Watt Hours Reactive',                   sDesc
ription := 'Elektrische Blindarbeit',                            sShortcut := 'Whr'),
  (*  eKilowattHoursReactive                 *) (sName := 'Kilowatt Hours Reactive',               sDesc
ription := 'Elektrische Blindarbeit',                            sShortcut := 'kWhr'),
  (*  eMegawattHoursReactive                 *) (sName := 'Megawatt Hours Reactive',               sDesc
ription := 'Elektrische Blindarbeit',                            sShortcut := 'MWhr'),
  (*  eMillimetersOfWater                    *) (sName := 'Millimeters of Water',                  sDesc
ription := 'Druck',                                              sShortcut := 'mmWS'),
  (*  ePer_Mille                             *) (sName := 'Per Mille',                             sDesc
ription := 'Promille',                                           sShortcut := '‰'),
  (*  eGrams_PerGram                         *) (sName := 'Grams per Gram',                        sDesc
ription := 'Massenanteil',                                       sShortcut := 'g/g'),
  (*  eKilograms_PerKilogram                 *) (sName := 'Kilograms per Kilogram',                sDesc
ription := 'Massenanteil',                                       sShortcut := 'kg/kg'),
  (*  eGrams_PerKilogram                     *) (sName := 'Grams per Kilogram',                    sDesc
ription := 'Massenanteil',                                       sShortcut := 'g/kg'),
  (*  eMilligrams_PeGram                     *) (sName := 'Milligrams per Gram',                   sDesc
ription := 'Massenanteil',                                       sShortcut := 'mg/g'),
  (*  eMilligrams_PeKilogram                 *) (sName := 'Milligrams per Kilogram',               sDesc
ription := 'Massenanteil',                                       sShortcut := 'mg/kg'),
  (*  eGrams_PerMilliliter                   *) (sName := 'Grams per Milliliter',                  sDesc
ription := 'Massenkonzentration',                                sShortcut := 'g/ml'),
  (*  eGrams_PerLiter                        *) (sName := 'Grams per Liter',                       sDesc
```

```
ription := 'Massenkonzentration',                                    sShortcut := 'g/l'),
  (*  eMilligrams_PerLiter                *) (sName := 'Milligrams per Liter',                sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'mg/l'),
  (*  eMicrograms_PerLiter                *) (sName := 'Micrograms per Liter',                sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'µg/l'),
  (*  eGrams_PerCubicMeter                *) (sName := 'Grams per Cubic Meter',               sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'g/m3'),
  (*  eMilligrams_PerCubicMeter           *) (sName := 'Milligrams per Cubic Meter',          sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'mg/m3'),
  (*  eMicrograms_PerCubicMeter           *) (sName := 'Micrograms per Cubic Meter',          sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'µg/m3'),
  (*  eNanograms_PerCubicMeter            *) (sName := 'Nanograms per Cubic Meter',           sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'ng/m3'),
  (*  eGrams_PerCubicCentimeter           *) (sName := 'Grams per Cubic Centimeter',          sDesc
ription := 'Massenkonzentration',                                    sShortcut := 'g/cm3'),
  (*  eBecquerels                         *) (sName := 'Becquerels',                          sDesc
ription := 'Aktivität (radioaktiver Stoff)',                         sShortcut := 'Bq'),
  (*  eKilobecquerels                     *) (sName := 'Kilobecquerels',                      sDesc
ription := 'Aktivität (radioaktiver Stoff)',                         sShortcut := 'kBq'),
  (*  eMegabecquerels                     *) (sName := 'Megabecquerels',                      sDesc
ription := 'Aktivität (radioaktiver Stoff)',                         sShortcut := 'MBq'),
  (*  eGray                               *) (sName := 'Gray',                                sDesc
ription := 'Energiedosis (ionisierende Strahlung)',                  sShortcut := 'Gy'),
  (*  eMilligray                          *) (sName := 'Milligray',                           sDesc
ription := 'Energiedosis (ionisierende Strahlung)',                  sShortcut := 'mGy'),
  (*  eMicrogray                          *) (sName := 'Microgray',                           sDesc
ription := 'Energiedosis (ionisierende Strahlung)',                  sShortcut := 'µGy'),
  (*  eSieverts                           *) (sName := 'Sieverts',                            sDesc
ription := 'Äquivalenzdosis (gewichtete Strahlendosis)',             sShortcut := 'Sv'),
  (*  eMillisieverts                      *) (sName := 'Millisieverts',                       sDesc
ription := 'Äquivalenzdosis (gewichtete Strahlendosis)',             sShortcut := 'mSv'),
  (*  eMicrosieverts                      *) (sName := 'Microsieverts',                       sDesc
ription := 'Äquivalenzdosis (gewichtete Strahlendosis)',             sShortcut := 'µSv'),
  (*  eMicrosievertsPerHour               *) (sName := 'Microsieverts per Hour',              sDescr
iption := 'Äquivalenzdosisleistung',                                 sShortcut := 'µSv/h'),
  (*  eDecibels_A                         *) (sName := 'Decibels A',                          sDesc
ription := 'A - bewerteter Schalldruckpegel',                        sShortcut := 'dB(a)'),
  (*  eNephelometric_TurbidityUnit        *) (sName := 'Nephelometric Turbidity Unit',        sDesc
ription := 'Trübung (Wasserqualität)',                               sShortcut := 'NTU'),
  (*  ePH                                 *) (sName := 'PH',                                  sDesc
ription := 'Wasserstoffionen-Konzentration',                         sShortcut := ''),
  (*  eGrams_PerSquareMeter               *) (sName := 'Grams per Square Meter',              sDesc
ription := 'Flächengewicht',                                         sShortcut := 'g/m2'),
  (*  eMinutes_PerDegreeKelvin            *) (sName := 'Minutes per Degree Kelvin',           sDesc
ription := 'Zeitänderung pro Temperatureinheit',                     sShortcut := 'min/K')
  ];

aMeasuringElement  : ARRAY[E_BA_MeasuringElement.First .. E_BA_MeasuringElement.Last] OF ST_BA_EnumI
nfo := [

  (*  eNI100                              *) (sName := 'NI100',                               sDesc
ription := 'Nickel 100',                                             sShortcut := ''),
  (*  eNI120                              *) (sName := 'NI120',                               sDesc
ription := 'Nickel 120',                                             sShortcut := ''),
  (*  eNI1000                             *) (sName := 'NI1000',                              sDesc
ription := 'Nickel 1000 (Standard)',                                 sShortcut := ''),
  (*  eNI1000_LS                          *) (sName := 'NI1000 LS',                           sDesc
ription := 'Nickel 1000 (Landis & Staefa)',                          sShortcut := ''),
  (*  eNTC1K8                             *) (sName := 'NTC1K8',                              sDesc
ription := 'NTC 1,8 kOhms',                                          sShortcut := ''),
  (*  eNTC1K8_TK                          *) (sName := 'NTC1K8 TK',                           sDesc
ription := 'NTC 1,8 kOhms (TK)',                                     sShortcut := ''),
  (*  eNTC2K2                             *) (sName := 'NTC2K2',                              sDesc
ription := 'NTC 2,2 kOhms',                                          sShortcut := ''),
  (*  eNTC3K                              *) (sName := 'NTC3K',                               sDesc
ription := 'NTC 3 kOhms',                                            sShortcut := ''),
  (*  eNTC5K                              *) (sName := 'NTC5K',                               sDesc
ription := 'NTC 5 kOhms',                                            sShortcut := ''),
  (*  eNTC10K                             *) (sName := 'NTC10K',                              sDesc
ription := 'NTC 10 kOhms',                                           sShortcut := ''),
  (*  eNTC10KPRE                          *) (sName := 'NTC10KPRE',                           sDesc
ription := 'NTC 10 kOhms (Precon)'                                   sShortcut := ''),
  (*  eNTC10K_3204                        *) (sName := 'NTC10K 3204',                         sDesc
ription := 'NTC 10 kOhms 3204',                                      sShortcut := ''),
  (*  eNTC10KTYP2                         *) (sName := 'NTC10K TYP2',                         sDesc
ription := 'NTC 10 kOhms TYP2',                                      sShortcut := ''),
  (*  eNTC10KTYP3                         *) (sName := 'NTC10K TYP3',                         sDesc
ription := 'NTC 10 kOhms TYP3',                                      sShortcut := ''),
  (*  eNTC10KDALE                         *) (sName := 'NTC10K DALE',                         sDesc
ription := 'NTC 10 kOhms DALE',                                      sShortcut := ''),
```

```
  (*  eNTC10K3A221                        *) (sName := 'NTC10K 3A221',                        sDesc
ription := 'NTC 10 kOhms 3A221',                                sShortcut := ''),
  (*  eNTC20K                             *) (sName := 'NTC20K',                              sDesc
ription := 'NTC 20 kOhms',                                      sShortcut := ''),
  (*  eNTC100K                            *) (sName := 'NTC100K',                             sDesc
ription := 'NTC 100 kOhms',                                     sShortcut := ''),
  (*  ePoti_Resolution_01                 *) (sName := 'Poti (Resolution 0,1 Ohms)',          sDesc
ription := 'Potentiometer (Resolution 0,1 Ohms)',               sShortcut := ''),
  (*  ePoti_Resolution_1                  *) (sName := 'Poti (Resolution 1 Ohm)',             sDesc
ription := 'Potentiometer (Resolution 1 Ohm)',                  sShortcut := ''),
  (*  eOutput_10_5000                     *) (sName := 'Output (10 to 5000 Ohms)',            sDesc
ription := 'Output (10 to 5000 Ohms)',                          sShortcut := ''),
  (*  eOutput_10_1200                     *) (sName := 'Output (10 to 1200 Ohms)',            sDesc
ription := 'Output (10 to 1200 Ohms)',                          sShortcut := ''),
  (*  ePT100                              *) (sName := 'PT100',                               sDesc
ription := 'PT 100',                                            sShortcut := ''),
  (*  ePT200                              *) (sName := 'PT200',                               sDesc
ription := 'PT 200',                                            sShortcut := ''),
  (*  ePT500                              *) (sName := 'PT500',                               sDesc
ription := 'PT 500',                                            sShortcut := ''),
  (*  ePT1000                             *) (sName := 'PT1000',                              sDesc
ription := 'PT 1000',                                           sShortcut := '')
  ];

aToggleMode    : ARRAY[E_BA_ToggleMode.First .. E_BA_ToggleMode.Last] OF ST_BA_EnumInfo := [

  (*  eSwitch                             *) (sName := 'Schalter',                            sDesc
ription := 'Einrastender Schalter',                             sShortcut := ''),
  (*  ePushButton                         *) (sName := 'Taster',                              sDesc
ription := 'Nicht-einrastender Schalter',                       sShortcut := '')
  ];

aAction      : ARRAY[E_BA_Action.First .. E_BA_Action.Last] OF ST_BA_EnumInfo := [

  (*  eDirect                             *) (sName := 'Direkt',                              sDesc
ription := 'Gleichläufiger Wirksinn',                           sShortcut := ''),
  (*  eReverse                            *) (sName := 'Indirekt',                            sDesc
ription := 'Gegenläufiger Wirksinn',                            sShortcut := '')
  ];

aEventState    : ARRAY[E_BA_EventState.First .. E_BA_EventState.Last] OF ST_BA_EnumInfo := [

  (*  eNormal                           *) (sName := 'Normal',                              sDesc
ription := '',                                                  sShortcut := ''),
  (*  eFault                            *) (sName := 'Fehler',                              sDesc
ription := '',                                                  sShortcut := ''),
  (*  eOffnormal                        *) (sName := 'Unnormal',                            sDesc
ription := '',                                                  sShortcut := ''),
  (*  eLowLimit                         *) (sName := 'Unter Grenzwert',                     sDesc
ription := 'Unterschreitung des definierten Grenzwert',         sShortcut := ''),
  (*  eHighLimit                        *) (sName := 'Über Grenzwert',                      sDesc
ription := 'Überschreitung des definierten Grenzwert',          sShortcut := '')
  ];

aReliability    : ARRAY[E_BA_Reliability.First .. E_BA_Reliability.Last] OF ST_BA_EnumInfo := [

  (*  eNoFaultDetected                  *) (sName := 'Kein Fehler',                         sDesc
ription := '',                                                  sShortcut := ''),
  (*  eNoSensor                         *) (sName := 'Sensorfehler',                        sDesc
ription := 'TODO',                                              sShortcut := ''),
  (*  eOverRange                        *) (sName := 'Über Bereich',                        sDesc
ription := 'Überschreitung des definierten Rohwert-Bereichs',  sShortcut := ''),
  (*  eUnderRange                       *) (sName := 'Unter Bereich',                       sDesc
ription := 'Unterschreitung des definierten Rohwert-Bereichs', sShortcut := ''),
  (*  eOpenLoop                         *) (sName := 'Drahtbruch',                          sDesc
ription := '',                                                  sShortcut := ''),
  (*  eShortedLoop                      *) (sName := 'Kurzschluß',                          sDesc
ription := '',                                                  sShortcut := ''),
  (*  eNoOutput                         *) (sName := 'Kein Ausgang',                        sDesc
ription := 'TODO',                                              sShortcut := ''),
  (*  eUnreliableOther                  *) (sName := 'Sonstiges',                           sDesc
ription := '',                                                  sShortcut := ''),
  (*  eProcessError                     *) (sName := 'Prozessfehler',                       sDesc
ription := 'TODO',                                              sShortcut := ''),
  (*  eMultiStateFault                  *) (sName := 'Multistate-
Fehler',               sDescription := 'TODO',                                          sShortcu
t := ''),
  (*  eConfigurationError               *) (sName := 'Konfigurationsfehler',                sDesc
ription := '',                                                  sShortcut := ''),
  (*              *) (),
```

```
  (*  eCommunicationFailure               *) (sName := 'Kommunikationsfehler',                 sDesc
ription := '',                                          sShortcut := ''),
  (*  eMemberFault                        *) (sName := 'Fehler Teilnehmer',                    sDesc
ription := 'TODO',                                      sShortcut := '')
  ];


aPolarity    : ARRAY[E_BA_Polarity.First .. E_BA_Polarity.Last] OF ST_BA_EnumInfo := [

  (*  eNormal                             *) (sName := 'Normal',                               sDesc
ription := 'Direkte Polarität',                         sShortcut := ''),
  (*  eReverse                            *) (sName := 'Invertiert',                           sDesc
ription := 'Indirekte Polarität',                       sShortcut := '')
  ];

aByteMappingMode  : ARRAY[E_BA_ByteMappingMode.First .. E_BA_ByteMappingMode.Last] OF ST_BA_EnumInfo
 := [

  (*  eIndex1N            *) (sName := 'Index 1-
N',                   sDescription := 'TODO',                                sShortcut := ''),
  (*  eBinary1N           *) (sName := 'Binary 1-
N',                    sDescription := 'TODO',                                 sShortcut := ''),
  (*  eIndexUpDown        *) (sName := 'Index Up-
Down',                  sDescription := 'TODO',                                 sShortcut := ''),
  (*  eBinaryUpDown       *) (sName := 'Binary Up-
Down',                sDescription := 'TODO',                                sShortcut := ''),
  (*  eBinaryDecimal      *) (sName := 'Binary Decimal',                       sDescription := 'TODO
',                                      sShortcut := '')
  ];

aPIDMode    : ARRAY[E_BA_PIDMode.First .. E_BA_PIDMode.Last] OF ST_BA_EnumInfo := [

  (*  eP1ID               *) (sName := 'P1ID',                              sDescription := 'Vorgelag
erter P-Anteil beeinflusst I- und D-Anteil', sShortcut := ''),
  (*  ePID                *) (sName := 'PID',                               sDescription := 'Addition
 von P-, I- und D-Anteilen (Standard)',      sShortcut := '')
  ];

aLoggingType    : ARRAY[E_BA_LoggingType.First .. E_BA_LoggingType.Last] OF ST_BA_EnumInfo := [

  (*  ePolled             *) (sName := 'Polled',                                  sDescription := 'Gen
eriert Log-Einträge in konstanten Intervallen',  sShortcut := ''),
  (*  eCOV                *) (sName := 'COV',                                     sDescription := 'Gen
eriert Log-Einträge bei Wertänderung',           sShortcut := ''),
  (*  eTriggered          *) (sName := 'Triggered',                               sDescription := 'Gen
eriert Log-Einträge ereignisgesteuert',          sShortcut := '')
  ];


aLanguage    : ARRAY[E_BA_Language.First .. E_BA_Language.Last] OF ST_BA_EnumInfo := [

  (*  eEnglish            *) (sName := 'Englisch',            sDescription := 'Englisch (1033)',
                              sShortcut := 'EN'),
  (*  eGerman             *) (sName := 'Deutsch',             sDescription := 'Deutsch (1031)',
                              sShortcut := 'DE')
  ];

END_VAR
```

More Information:
www.beckhoff.com/te1000/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com