FAQ COMBIVIS studio 6

Software Engineering Guideline for IEC 61131-3 libraries and application FAQ No.

Part	Version	Revision	Date	Status
en	6.4.0.1	000	2022-08-04	Released

History:

Date	Version	Comment	Author
2011_10_18	2.0.	Created for generation 6	Fischer
2011_11_09	2.1	Review and Release	Kaiser, Grabbe, Pieper, Fischer
2013_03_19	2.2	Review	CCT EUROPE
2017_11_20	6.4.0.0	- KEB Automation KG standards introduced - Library docformat restructured text introduced - Library categories chapter updated	Fischer
2022_08_04	6.4.0.1	- Changed library history style	Schneider

KEB Automation KG
Südstraße 38
32683 Barntrup, Germany
E-mail: info@keb.de
Tel. +49 5263 401-0
Fax +49 5263 401-116
www.keb.de

District Court Lemgo HRA 5649
DUNS-No. 314108728
VAT-No. DE309087075
Bank Details: Sparkasse Paderborn-Detmold
IBAN DE 19 4765 0130 0000 0060 07
BIC WELADE3L

General Partner:
Vittorio Tavella
KEB Verwaltungs-GmbH, Barntrup
District Court: Lemgo HRB 8965
Directors: Curt Bauer CMO,
Ralf Lutter COO, Vittorio Tavella CFO,
Wolfgang Wiele CTO

FAQ COMBIVIS studio 6

Content

Introduction

This document shows the guidelines for KEB IEC 61131-3 libraries and applications based on the IDE KEB COMBIVIS studio 6.
These guidelines are meant to realise a worldwide uniform look of IEC 61131-3 libraries, projects and applications.
Programing close to these guidelines means:

- Create reusable IEC 61131-3 code.
- Reduce the time for application development and maintenance.
- Enhance the collection of ready-to-use function modules for your application sector.
- Benefit from the global KEB application know-how.

General Guidelines

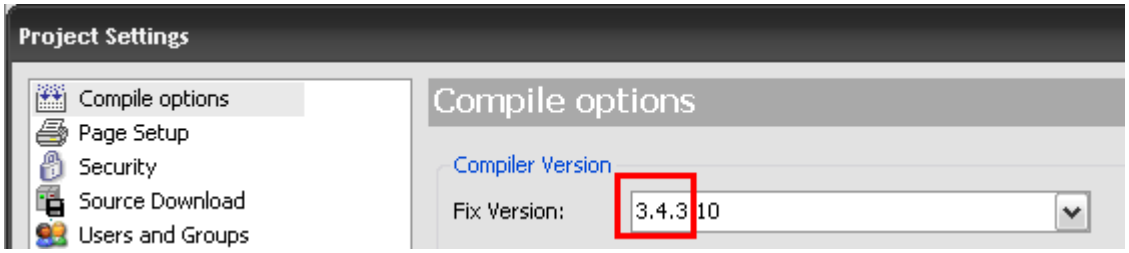
Project Name

The project names are formed at least by the application name and a version code. Additionally the company name and type of control can be appended.

- The name shall only contain the characters 0..9, letters A..Z, a..z and underscores
- Dots are allowed for the version tag only.

<Company>_<Application>_<Target Control(optional)>_<Version>
E.g.: KEB_Winder_C6C_3.4.3.15

<Version>x.x.x.x = <Compiler Version>x.x.x.x + <Implementation Version>
The compiler version can be appended as version code, additional the implementation version is counted up at each change. This version code is also used for the internal project information and for entries in the version history text file.
E.g.: 3.4.3.15 Means: Compiler V3.4.3.10, 5th implementation.



<Target Control>
Contains the name of the destination control.
C6x: KEB COMBICONTROL C6 device
C6Cx: COMPACT class version x
C6Sx: SMART class version x
C6Ex: ECONOMY class version x
C6Px: PERFORMANCE class version x
P6x: P6 control unit version x
H6x: H6 control unit version x
WIN: Windows simulation device

IEC Languages

The following table shows in what cases which language may be used.

KEB Object	Programming language
State machines (in project)	ST
Function blocks, Libraries	ST
Samples	ST, FBD, CFC, SFC

Name Conventions

To be used in applications and new libraries.

- **Names always have to be written in English!**
- Allowed characters in general: A..Z, a..z, 0..9, "_"
- Do not use unknown/ misleading abbreviations, see abbreviation list. Longer, meaningful names are preferred.

Camel Case Notation

The first **letter** of each **word** of a base name should be a capital letter, the others should be small ones (So called "CamelCase"). The first character must always be an upper case letter (A-Z) except for special prefixes, like p for pointer. Underscores are to be avoided.
E.g.: MyCamelCaseVariable

The following tables give an image of the "special cases":

POU	Sample	Prefix
PROGRAM	**FileHandling()**; **FileHandling**.Done **HmiCom(Enable:=TRUE)**;	
FUNCTION	**CalcDiameter(Radius)**;	
FUNCTION BLOCK **(Class name)**	*Prefix KEB_ for KEB libraries (For internal/ help FBs the company prefix can be omitted).* **KEB_HomingOnBlock** **MC_ReadActualPosition** **MC_MoveVelocity**	

FUNCTION BLOCK
 	
(Instance name)
	Hsp5Master: KEB_Hsp5Master;
	DataLoggerInverter: DataLogger;
	DataLoggerPlc: DataLogger;
	
	For well-known FBs, the class name is possible as instance prefix.
	TonWait: TON; //wait timer
	TonInit: TON; //init timer
	RtrigReset: R_trig; //trigger positive edge of reset input	

	Variables, properties, methods and actions are handled in the same way. ***No prefix is used. No Underscores. Start with upper case letter.***	
ACTION	MyPid.**ResetIntegral()**; //reset Integral part of PID controller	
PROPERTY	PropValueOld:= LoggerPitch**.LastIndex**; LoggerPitch**.LastIndex** := PropValueNew;	
METHOD	CleanUpDone := LoggerPitch**.CleanBuffer()**;	

VAR **VAR_IN_OUT** **VAR INPUT** **VAR OUTPUT**	*Correct:* *Wrong:* **MaxFileSize** max_file_size **SetVelocity** setVelo **ActPositionInc** CurPosIncre //Actual position [increments] **ActDcVoltage** actDC_Voltage, act_DC_Voltage **LogData: ARRAY [0..10] OF REAL;**	
VAR GLOBAL (Access)	*Always use name of global variable list to access global variables.* GVL_MOTOR.**ActCurrent**	
Loop counter	**i, j, k, l, m, n** *(one letter)*	
Pointers	**pBuffer**: POINTER TO BYTE;	**p<>**
INTERFACE	**IKebComSlave** *(Prefix is 'capital i' without underscore)*	**I<>**
Visualization object	**VISU_Master, VISU_Home, VISU_ErrorHandling**	**VISU_<>**
Visualization library template	**KEB_VISU_ChannelHandler** **KEB_VISU_DriveControl**	**KEB_VISU <>**

Upper Case Notation

Static definitions like data unit types and constants will always be written in upper case letters. Parts of names will be separated by underscore "_".

E.g.: TO_BE_CARVED_IN_STONE

POU	Sample	Prefix
VAR CONSTANT	**LAST_INDEX**: WORD := 20; **PI**: REAL := 3.14159; **HSP5_MAX_BUFF_CNT**: WORD := 64;	
ENUM	*The type definition consists of prefix + upper case notation. Members of an enumeration are handled like constants.* TYPE **T_KEB_SAMPLE_ERROR** : (NO_ERROR:= 0, //no active error TIMEOUT:=1, //timeout error INTERNAL:= 2, //internal FB error INVALID_DATA:= 3 //invalid data error); END_TYPE	**T_<>**
STRUCT	*The type definition consists of prefix + upper case notation. Members of a structure are handled like variables (CamelCase), as their values are not constant.* TYPE **T_LOG_ENTRY** : STRUCT	**T_<>**

POU	Sample	Prefix

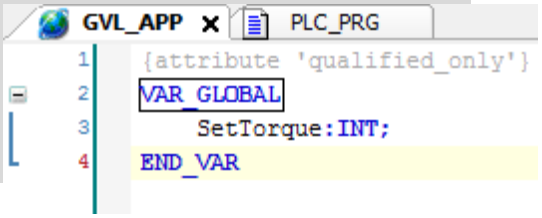
	ErrorID: T_KEB_SAMPLE_ERROR; //error identification
	LogTime: DWORD; //error time stamp
	END_STRUCT
	END_TYPE
	
	Instances of structures are handled like variables (CamelCase).
	
	LogData: ARRAY [0..99] OF **T_LOG_ENTRY;**	

Global variable list	**GVL_GATEWAY**	**GVL_**

Hint: For ENUMs and GVLs the pragma {attribute 'qualified_only'} shall be used in the top of the declaration to enforce the access via the GVL name.

```
{attribute 'qualified_only'}
VAR_GLOBAL
        SetTorque :REAL;
END_VAR

Sample instruction: GVL_APP.SetTorque := 100;
```



Recommended Numeric Data Types

To minimize the need of type conversions use the recommended basic data types.

E.g.:
Use BYTE instead of UNSIGNED SHORT INT (USINT).
Use WORD instead of UNSIGNED INT (UINT).
Use DWORD instead of UNSIGNED DINT (UDINT).

Data type	Lower limit	Upper limit	Size
BOOL	**FALSE**	**TRUE**	**8 Bit**
BYTE	**0**	**255**	**8 Bit**
WORD	**0**	**65 535**	**16 Bit**
DWORD	**0**	**4 294 967 295**	**32 Bit**
LWORD	**0**	**2^{64}-1**	**64 Bit**
SINT	**-128**	**127**	**8 Bit**
INT	**-32 768**	**32 767**	**16 Bit**
DINT	**-2 147 483 648**	**2 147 483 647**	**32 Bit**
LINT	**-2^{63}**	**2^{63}-1**	**64 Bit**
REAL	**1.175494351e-38 to 3.402823466e+38**		**32 Bit**
LREAL	**2.2250738585072014e-308 to 1.7976931348623158e+308**		**64 Bit**

Speaking POU & Variables names

For each variable a meaningful, speaking description should be found.

Type	Rule	Sample
Method **Action**	*Methods starts with a verb.*	GetPosition() ResetError()
BOOL	*Bool names describes the action/ status of the TRUE state.*	EnableVoltage //switch StartHoming VoltageEnabled //status TorqueControlActive InitDone

Standard Identifier

In general, speaking names are to be used.
For common terms a uniform abbreviation or standard term shall be used.

Term	Abbreviation
Application	
Actual Value	Act
Setpoint	Set
Value from last cycle	Old
Minimum	Min
Maximum	Max
Position	Pos
Increments	Inc
Velocity	Vel
Frequency	Frq
Acceleration	Acc
Deceleration	Dec
Absolute	Abs
Relative	Rel

Term	Abbreviation
Communication	
Process data	PD
Address	Addr
Request	Req
Response	Rsp
Index	Idx
Buffer	Buff
Count	Cnt
Memory	Mem
Length	Len
Identifier	Id
Source	Src
Destination	Dest

Formatting

Per level of function depth an indention of 1 tabulator is used.

Sample IF Instruction

```
IF (x>0) THEN
  BuffCnt := BuffCnt + BuffOffset;  //Count up
ELSIF (x<0) THEN
  BuffCnt := BuffCnt - BuffOffset; //Count down
ELSE
  Error:=TRUE;       //Set error
END_IF
```

Sample FOR Instruction

```
FOR i:= 1 TO AXIS_CNT DO
  DriveCtrl[i](Start:=TRUE);  //Start all drives
END_FOR
```

Sample CASE OF Instruction

```
//main state machine
CASE State OF

  STATE_NOT_ENABLED: //FB not active, trigger Execute input to start action
  (*================================================================*)
    IF Execute THEN
      State:=STATE_BUSY; //switch to busy state
      Busy:=TRUE;
      IsBusy(); //work on the FB task
    END_IF

  STATE_BUSY://FB is busy
  (*================================================================*)
    IsBusy(); //work on the FB task

  STATE_DONE://action sucessfully done, FB is idle, reset Execute input to reset outputs
  (*================================================================*)
    IF NOT Execute THEN
      ResetOutputs();
    END_IF

  STATE_ERROR: // error occured, FB is idle, reset Execute input to reset outputs
  (*================================================================*)
    IF NOT Execute THEN
      ResetOutputs();
    END_IF

  ELSE //unknown state, should not happen
  (*================================================================*)
    SetError(ErrID:=999); //set unknown error

END_CASE
```

Documentation/ Comments

General Rules

- **Comments always have to be written in English!**
- Every public variable declaration needs a comment!
- Comments describe the function of a POU/ code line, not the instruction!

Good sample:
```
IF (x > 100) THEN // If max. buffer index reached
    x := 0;       //  reset buffer index
END_IF
```

Bad sample:
```
IF (x > 100) THEN  // If x > 100
    x := 0;      //  reset x to 0
END_IF
```

- Comments over more than one line are allowed (`(*  *)`).
- Eliminate not used, out-commented code for more readability.
- Give detailed information if the source code is not a standard solution,
- e.g.: //Attention, this is a workaround for case x; //This is a test case.
- Give detailed information if something is missing, e.g.: //To Do: check for limits

- **Tip: Message pragmas can be created to inform the user during compilation about critical things (See CODESYS online documentation).**

Pragma	Message type
{text 'textstring'}	**Text**: The specified **textstring** will be displayed.
{info 'textstring'}	**Information** The specified **textstring** will be displayed.
{warning digit 'textstring'}	**Warning** The specified **textstring** will be displayed.
{error 'textstring'}	**Error** The specified textstring will be displayed.

Function Block Rules

- An English POU description has to be added ABOVE the first keyword (E.g. FUNCTION BLOCK, see sample description below). This POU documentation will be available for end- users in the online help and in the library manager view.
- You can insert a change history for each POU after the first keyword. This text is NOT published in the online help and in the library manager view. (See sample description below).
- Author: Full name has to be used (E.g.: Hugo Mueller).
- Date format is: year_month_day (E.g.: 2011_02_03).

Special Documentation Commands

The markup language reStructuredText can be used to prepare a well-formated documentation inside the sourcecode. The documentation itself will be generated automatically, when the library is imported into the ide.
Linebreaks are detected automatically. Historical commands like ~~
~~, ~~<p/>~~ are to be avoided.
The feature has to be activated in the project properties for each library, see 0.

Sample Function Block Description

```
(*
This function block (FB) controls a KEB F5 Drive (A-board) in different
drive modes without softmotion.

Drivemodes: VELOCITY, POSITIONINGABSOLUTE,
 POSITIONINGRELATIVE, SETPOSITION, HOMING
*)

FUNCTION_BLOCK KEB_DriveCtrl
(*
LibVersion      Date                Author              Topic
3.4.0.0  2010_06_09          Hugo Mueller        generated
3.4.0.1  2011_02_03          Hugo Mueller        New input x
*)

VAR_INPUT
        Execute:BOOL; //Positive edge enables FB
END_VAR

VAR_OUTPUT
        Done:BOOL; //TRUE: Successfully done
        Busy:BOOL;      //TRUE: FB is working
        Error:BOOL;     //TRUE: Error occured
        ErrorID:INT;    //Shows error ID
END_VAR
```

Sample documentation

Start of FB

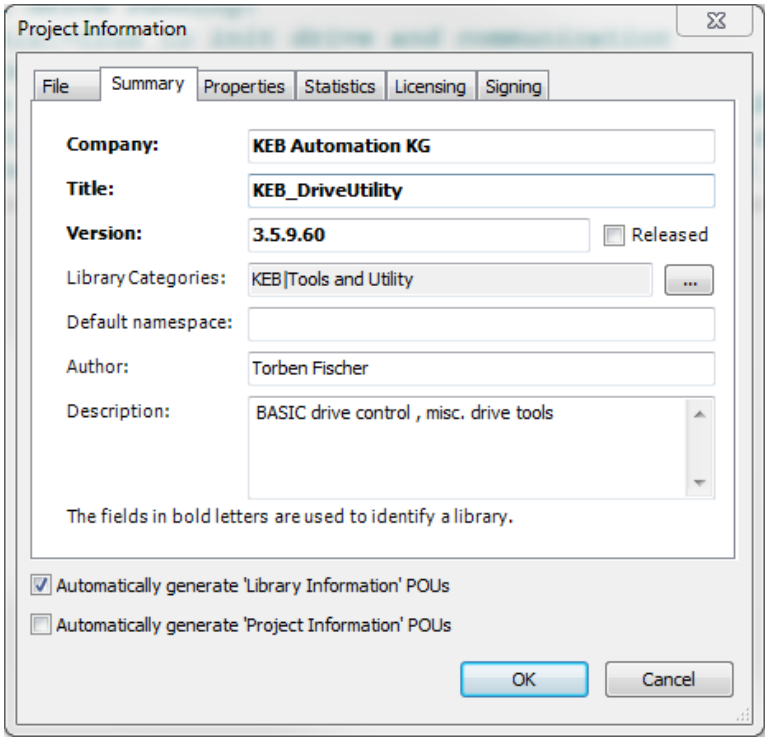
Hidden version history

Start of declaration part

Special Library Guidelines

Project Information

For libraries some strict rules for the project information have to be followed.



Company: Use the correct term of the company, e.g. „KEB Automation KG"

Title: The library name starts with „KEB_" + CamelCase notation

Version: Use the version code of the current compiler version. The last number is counted up for every new compiled library version (so called implementation version).
Do not forget to count up the version each time a new compiled library is created!

Library Categories: The categories are fixed inside the file "KebLibraryCategory.libcat.xml" (see IDE installation path); E.g.: „KEB|Tools and Utility"

Library categories have to be requested from the KEB Automation KG, Barntrup.
Usage of the head category "KEB|" is forbidden, a library has always to be placed into a sub-category.

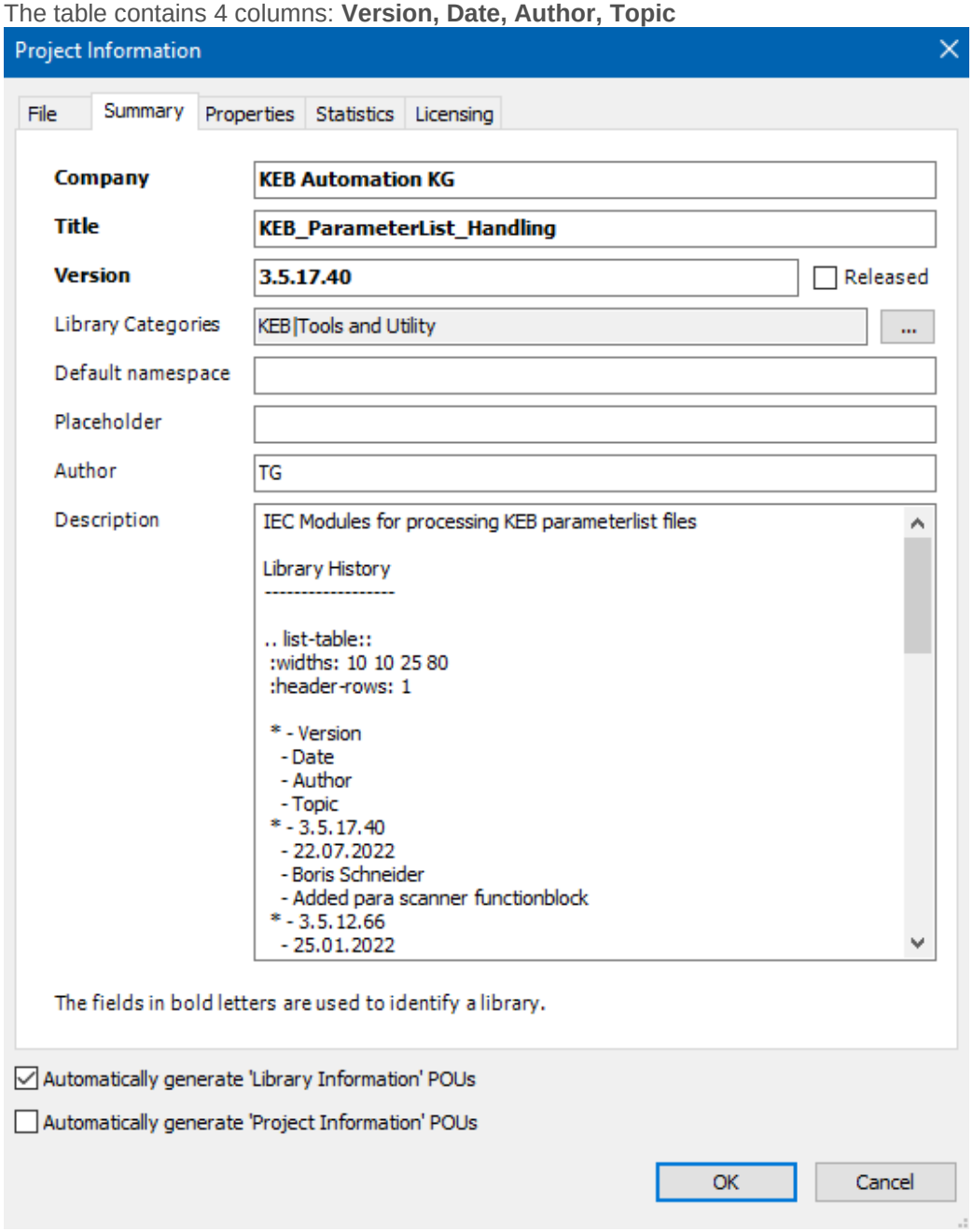
Author: Author of the library, full name.

Description: Short library description, same as in the version history text file.

Automatically generate 'Library Information' POUs: Option has to be enabled.

Library History

To document all library/project changes, a table with the header “Library History” is embed into the Project Information/Description. Use the ReStruct object “list-table”.
The table contains 4 columns: **Version, Date, Author, Topic**



Sample of representation inside the library manager

Library History

Version	Date	Author	Topic
3.5.17.40	22.07.2022	Boris Schneider	Added para scanner functionblock
3.5.12.66	25.01.2022	Boris Schneider	Fixed a bug that caused the first parameter not to be downloaded correctly
3.5.12.65	26.11.2021	Boris Schneider	Fixed a limitation for 64-Bit compatibility
3.5.12.64	04.08.2020	Boris Schneider	#38052 Fixed a bug regarding reinit of the lists
3.5.12.63	29.05.2020	Boris Schneider	#37706 Changed download rules for WA parameter to follow the dw5 guideline #37286 Fixed reinit issue #37019 Fixed issue with not resetable errors #37018 Added repeat/ignore counter to output #36350 encrypted library source files

Sample of Library History text

```
This is the sample library with no function at all


Library History
----------------

.. list-table::
 :widths: 10 10 25 80
 :header-rows: 1

 * - Version
   - Date
   - Author
   - Topic
 * - 3.5.17.40
   - 22.07.2022
   - Boris Schneider
   - Added para scanner functionblock
* - 3.5.12.64
   - 04.08.2020
   - Boris Schneider
   - #38052 Fixed a bug regarding reinit of the lists
 * - 3.5.12.63
   - 29.05.2020
   - Boris Schneider
   - | #37706 Changed download rules for WA parameter to follow the dw5 guidel
     | #37286 Fixed reinit issue
     | #37019 Fixed issue with not resetable errors
     | #37018 Added repeat/ignore counter to output
     | #36350 encrypted library source files
```

Library description

Header of Library History

Definition of the restruct text object

Header of columns

Table content (standard)

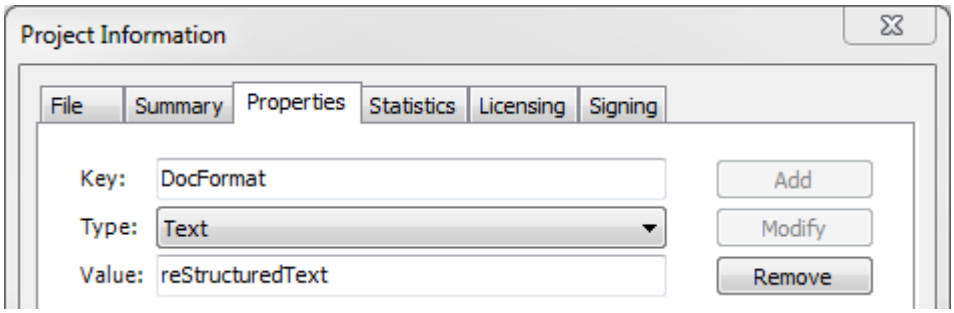
Table content (ticket reference)

Table content (multible lines)

Project Information Properties

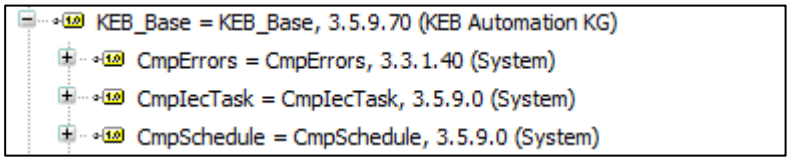
The library documentation has to be set to “restructuredText” by adding the following key:

Key: DocFormat
Type: Text
Value: reStructuredText

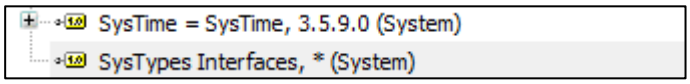


Library Manager

It is mandatory to add all libraries **using placeholders**!
This will catch a valid set of libraries depending on the compiler version and device.



Usage of the asterix- placeholder (always newest library installed.) is not allowed, except for interface libraries.

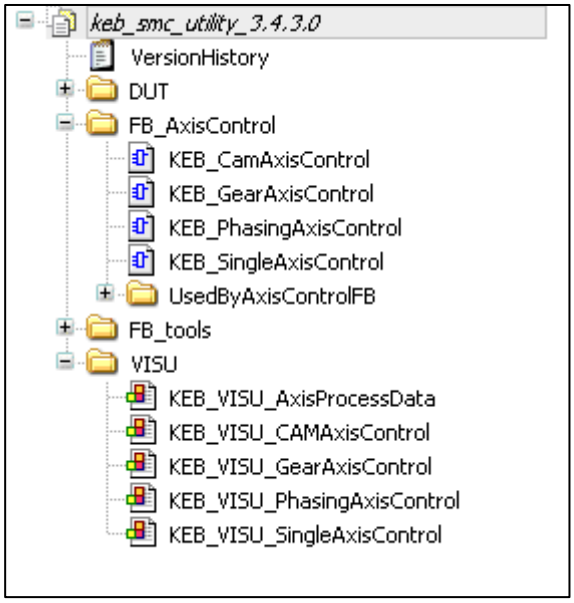


POU Organization

Use the **POU view** (Alt+0) (Instead of “Navigator-View”), to organize all POUs in a clearly arranged way.

Use subfolders to sum up elements belonging to each other.
Use subfolders if there is more than one POU of any type.
This is important for the automatic online documentation of the library!

Sample Subfolders:
DUT : Data Unit Types
FB: Function Blocks
VISU: Visualization Templates
Help/ Intern/ Base: Help POUs used by other POUs, not important for the end user.



Visualization Templates

Names of visualization templates have to begin with:

<KEB_VISU_>
E.g.: KEB_VISU_MotorMonitoring

Library Warnings

All warnings have to be eliminated. If a warning cannot be solved by source code correction, a pragma can used instead.
E.g.: {warning disable C0371} … {warning restore C0371

Annex

Best Practice Tips

Take care about index errors in loops and arrays: Index should be >=0 and <= limit.
Take care about division by zero: Such case needs an error handling or limitation.
Check every variable, if overflow/ underflow is possible.
Always check the POU feedback for errors/ timeouts.
Time critical processes: Really every line of code should be kept as small as possible.
Control structures like IF/ THEN/ ELSE or CASE OF should have a branch when a situation comes up which was not planned: E.g. CASE OF instructions should have an ELSE branch which contains an error code message.
No multi-purpose functions/procedures, prefer smaller separate subroutines.
Task Management: • Keep it simple, as few tasks as possible. • Check for cycle time overflows after a whole application cycle. • Choose an appropriate cycle time for the technical process, make sure that the CPU has enough idle time to guarantee deterministic behavior.
Naming: • Do not use an object name twice if the access right is global. • Do not use object names which can be mistaken, avoid combinations of similar letters and numbers, e.g. IO, 1O, i0. • Do not use unpronounceable names. A longer meaningful name is always the better idea. E.g. ResetInverter is a better name than RstInv.
Programming: Avoid “Magic numbers”: Declare a constant variable instead of using a static number somewhere in the source code. This makes maintenance much easier, e.g. in case the number changes in the future. **Bad:** FOR i := 1 TO 5 DO (drive[i].SetSpeed := 100); END_IF **Good:** LAST_DRIVE: INT := 5; SET_SPEED: INT:= 100; FOR i := 1 TO LAST_DRIVE DO (drive[i].SetSpeed := SET_SPEED); END_IF

Useful Pragmas

For a complete list, see the Online help.

Pragma/ Syntax	Description
{text 'textstring'}	Text: The specified textstring will be displayed.
{info 'textstring'}	Information The specified textstring will be displayed.
{warning digit 'textstring'}	Warning The specified textstring will be displayed.
{error 'textstring'}	Error The specified textstring will be displayed.
{attribute 'hide'}	Prevent variables, methods, etc. or even whole signatures from being displayed within the "List Components" functionality, the input assistant or the declaration part in online-modus. Only the variable subsequent to the pragma will be hidden. Use this attribute e.g. for private methods.
{attribute 'hide_all_locals'}	Prevent all local variables, methods, etc. or even whole signatures from being displayed within the "List Components" functionality, the input assistant or the declaration part in online-modus.
{attribute 'qualified_only'}	Assign on top of a global variable list or enumeration declaration. The variables of this list can only be accessed by using the global variable name/ enumeration name, e.g. GVL_MOTOR.ActCurrent. This is useful to avoid name mismatch with local variables.
{warning disable C0XYZ} … {warning restore C0XYZ	The code inbetween these pragmas will not trigger the warning XYZ.

FAQ COMBIVIS studio 6

Disclaimer

KEB Automation KG reserves the right to change/adapt specifications and technical data without prior notification. The safety and warning reference specified in this manual is not exhaustive. Although the manual and the information contained in it is made with care, KEB does not accept responsibility for misprint or other errors or resulting damages. The marks and product names are trademarks or registered trademarks of the respective title owners.

The information contained in the technical documentation, as well as any user-specific advice in verbal or in written form are made to the best of our knowledge and information about the application. However, they are considered for information only without responsibility. This also applies to any violation of industrial property rights of a third-party.

Inspection of our units in view of their suitability for the intended use must be done generally by the user. Inspections are particular necessary, if changes are executed, which serve for the further development or adaption of our products to the applications (hardware, software or download lists). Inspections must be repeated completely, even if only parts of hardware, software or download lists are modified.

Application and use of our units in the target products is outside of our control and therefore lies exclusively in the area of responsibility of the user.

KEB Automation KG
Südstraße 38 • D-32683 Barntrup
fon: +49 5263 401-0 • fax: +49 5263 401-116
net: www.keb.de • mail: info@keb.de