DOCUMENTATION ISG-kernel

Functional description
Export V.E. variables

Short description:
FCT-C22

www.isg-stuttgart.de
support@isg-stuttgart.de

Documentation version: 1.24
04/06/2025

Preface

Legal information

This documentation was produced with utmost care. The products and scope of functions described are under continuous development. We reserve the right to revise and amend the documentation at any time and without prior notice.

No claims may be made for products which have already been delivered if such claims are based on the specifications, figures and descriptions contained in this documentation.

Personnel qualifications

This description is solely intended for skilled technicians who were trained in control, automation and drive systems and who are familiar with the applicable standards, the relevant documentation and the machining application.

It is absolutely vital to refer to this documentation, the instructions below and the explanations to carry out installation and commissioning work. Skilled technicians are under the obligation to use the documentation duly published for every installation and commissioning operation.

Skilled technicians must ensure that the application or use of the products described fulfil all safety requirements including all applicable laws, regulations, provisions and standards.

Further information

Links below (DE)

https://www.isg-stuttgart.de/produkte/softwareprodukte/isg-kernel/dokumente-und-downloads

or (EN)

https://www.isg-stuttgart.de/en/products/softwareproducts/isg-kernel/documents-and-downloads

contains further information on messages generated in the NC kernel, online help, PLC libraries, tools, etc. in addition to the current documentation.

Disclaimer

It is forbidden to make any changes to the software configuration which are not contained in the options described in this documentation.

Trade marks and patents

ISG®, ISG kernel®, ISG virtuos®, ISG dirigent®, TwinStore® and the associated logos are registered and licensed trade marks of ISG Industrielle Steuerungstechnik GmbH.

The use of other trade marks or logos contained in this documentation by third parties may result in a violation of the rights of the respective trade mark owners.

Copyright

General and safety instructions

Icons used and their meanings

This documentation uses the following icons next to the safety instruction and the associated text. Please read the (safety) instructions carefully and comply with them at all times.

Icons in explanatory text

➢ Indicates an action.

⇨ Indicates an action statement.

⚠ DANGER

Acute danger to life!

If you fail to comply with the safety instruction next to this icon, there is immediate danger to human life and health.

⚠ CAUTION

Personal injury and damage to machines!

If you fail to comply with the safety instruction next to this icon, it may result in personal injury or damage to machines.

Attention

Restriction or error

This icon describes restrictions or warns of errors.

Notice

Tips and other notes

This icon indicates information to assist in general understanding or to provide additional information.

Example

General example

Example that clarifies the text.

Programing Example

NC programming example

Programming example (complete NC program or program sequence) of the described function or NC command.

Release Note

Specific version information

Optional or restricted function. The availability of this function depends on the configuration and the scope of the version.

Table of contents

1 Overview

Task

The export functionality generates a channel-specific data structure containing all variables from an existing “List of External Variables” (referred to as V.E List or Variable) of a machine configuration.

Characteristics

This generated data structure can be imported to a PLC environment, thus permitting the PLC to access the V.E variables. It also permits the rapid and reliable creation of an interface between the NC controller and the PLC for data transfer.

Parametrisation

The parameter P-EXTV-00022 defines the specified number of characters is used for string variables.

Programming

The data structure is exported to a small NC program by the #EXPORT VE[...] [▶ 6] command Since the configuration of V.E variables no longer changes after start-up, this operation is usually executed only once when the machine is started.

Mandatory note on references to other documents

For the sake of clarity, links to other documents and parameters are abbreviated, e.g. [PROG] for the Programming Manual or P-AXIS-00001 for an axis parameter.

For technical reasons, these links only function in the Online Help (HTML5, CHM) but not in pdf files since pdfs do not support cross-linking.

2 Description

Data transfer between PLC and CNC via V.E variables

V.E variables permit the transfer of data in any direction between an NC program and the PLC.

The PLC can access V.E variables by simulating them as a data structure in the PLC.

Initial situation

A variable list of the configuration is created.

2.1 Generating the output file (#EXPORT VE)

The NC command **#EXPORT VE [..]** generates the required data structure for the V.E. variables for the channel in which the command is used.

With multi-channel systems, the NC command must be used in each channel in order to generate the data structure for the particular channel.

Notice

The identifier <i> in the filename of the output file is a placeholder for the CNC channel number.

Syntax:

#EXPORT VE [3S | **KW]**

3S	For 3S CODESYS 2.3 PLC environment:
	Output file: plc_3s_ve_types_ch_**<i>**.exp
	For 3S CODESYS 3.5 PLC environment:
	Output file: plc_3s_ve_types_ch_**<i>**.xml
KW	For MULTIPROG PLC environment:
	Output file: plc_kw_ve_types_ch_**<i>**.exp

The output directory is defined by P-STUP-00020 or as of V3.1.3052.05 by P-CHAN-00403.

Syntax:

#EXPORT VE [3S | **TWINCAT** | **KW]**

3S / TWINCAT	For TwinCAT 2 and the original 3S CODESYS 2.3 PLC environment:
	Output file: plc_3s_ve_types_ch_**<i>**.exp
	For TwinCAT 3 and the original 3S CODESYS 3.5 PLC environment:
	Output file: plc_3s_ve_types_ch_**<i>**.xml
	Output directory:
	• For TwinCAT, the output path defined in the SystemManager is used to search for NC programs. See: CNC configuration - CNC task GEO - HLI tab HLI - entry box: NC file path
	• For 3S: Directory specified by P-STUP-00020 or as of V3.1.3052.05 by P-CHAN-00403
KW	For MULTIPROG PLC environment:
	Output file: plc_kw_ve_types_ch_**<i>**.exp
	Output directory: application-specific (P-STUP-00020 or using P-CHAN-00403) as of V3.1.3052.05 and higher

If no output directory is specified in a TwinCAT configuration, the output file is placed in the following directory depending on the TwinCAT version:

- TwinCAT 2 32-bit: Main directory C:\
- TwinCAT 3 64-bit: C:\Windows\SysWOW64

This is dependent on the corresponding write authorisations in each directory.

Notice

The call of the CNC command #EXPORT VE absolutely requires the specification of the PLC destination system as parameter. The result is named accordingly.

An error message 20509 is output if the parameter is missing.

Programing Example

Generate the output file

```
#EXPORT VE [TWINCAT] ;Generate V.E. PLC structure for TwinCAT
#EXPORT VE [3S] ;Generate V.E. PLC structure for 3S CODESYS
#EXPORT VE [KW] ; Generate V.E. PLC structure for MULTIPROG from KW
```

The command can be placed in an NC program or can be executed as a manual block. The command generates a file which is declared in a data structure compliant with IEC 61131-3 for all V.E variables created in the NC channel.

The generated file corresponds to the import/export format for the CODESYS or MULTIPROG development environments and can be imported there directly.

Notice

Additional structure declarations are required in the output file.

2.2 Errors on exporting

The declaration of the V.E variables is checked before the function generates the PLC data structure.

Any error messages occurring are logged in the (EXPORT) output file.

3 Examples

The two examples below show how the exported PLC structure is integrated into a PLC project starting from a V.E. variable list.

The two examples only differ in the parameters defined in P-EXTV-00022. This parameter determines the length of variables of the string type. The length of the type has impacts on the memory layout generated.

3.1 Example 1- Use short text strings

3.1.1 V.E. variable list

Assignment in V.E. variable list:

```
#
use_extended_string_var      0     # P-EXTV-00022
#
anzahl_belegt                4
#
var[0].name                  FARBE
var[0].type                  UNS16
var[0].scope                 CHANNEL
var[0].synchronisation       FALSE
var[0].access_rights         READ_WRITE
var[0].array_elements        3
#
var[1].name                  TEXT
var[1].type                  STRING
var[1].scope                 CHANNEL
var[1].synchronisation       FALSE
var[1].access_rights         READ_WRITE
var[1].array_elements        2
#
var[2].name                  INFO_IN
var[2].type                  OFFSET
var[2].scope                 GLOBAL
var[2].synchronisation       FALSE
var[2].access_rights         READ_WRITE
var[2].array_elements        2
#
var[3].name                  INFO_OUT
var[3].type                  OFFSET
var[3].scope                 GLOBAL
var[3].synchronisation       FALSE
var[3].access_rights         READ_WRITE
var[3].array_elements        2
var[3].create_hmi_interface  0
#
struct[0].name               OFFSET
struct[0].element[0].name    X
struct[0].element[0].type    UNS16
struct[0].element[1].name    Y
struct[0].element[1].type    UNS16
#
```

3.1.2 Example of output file for CODESYS

Representation in the exported file:

```
TYPE STRING_20:
STRUCT
    token:STRING(20);
    f1_st: ARRAY[0..2] OF BYTE;
END_STRUCT
END_TYPE

TYPE STRING_20_2:
STRUCT
    token:STRING(20);
    f1_st: ARRAY[0..106] OF BYTE;
END_STRUCT
END_TYPE

TYPE OFFSET:
STRUCT
    X: UINT;
    Y: UINT;
END_STRUCT
END_TYPE

TYPE VE_CHANNEL_DATA_CH_1:
STRUCT
    FARBE: ARRAY[0..2] OF UINT;
    f1: ARRAY[0..17] OF BYTE;
    TEXT: ARRAY[0..1] OF STRING_20;
END_STRUCT
END_TYPE

TYPE VE_GLOBAL_DATA_FROM_CH_1:
STRUCT
    INFO_IN: ARRAY[0..1] OF OFFSET;
    INFO_OUT: ARRAY[0..1] OF OFFSET;
END_STRUCT
END_TYPE
```

3.1.3 PLC example

Integrating the structure in a 3S PLC program:

```
VAR
  (* Use generated structure descriptions *)
  p_ve_chan_1 : POINTER TO VE_CHANNEL_DATA_CH_1;
  p_ve_glob   : POINTER TO VE_GLOBAL_DATA_FROM_CH_1;
  text        : STRING(20);
  init_ve_ptr : BOOL := TRUE;

END_VAR

(* Ensure that the internal management data is initialised *)

Hli(Start := TRUE);


IF Hli.Initialised = TRUE AND Hli.Error = FALSE THEN

  IF init_ve_ptr = TRUE THEN

    (* Provide pointer to structure(s) *)
    p_ve_chan_1 := ADR( gpVECH[0]^.ext_var32[0]);
    p_ve_glob   := ADR(gpVEGlobal^.ext_var32[0]);

  END_IF;

  (* Work with the variables (read, write) *)
  text := p_ve_chan_1^.TEXT[0].token;
  p_ve_chan_1^.FARBE[1] := 2;
END_IF
```

3.1.4 Example of output file for MULTIPROG

The following export for MULTIPROG is based on the identical V.E variable list [▶ 8] as the export for CODESYS [▶ 9].

```
TYPE
  TYPE_STRING_20 : ARRAY[0..20] OF BYTE;
END_TYPE

TYPE
  ALIGN_STRING_20_1 : ARRAY[0..2] OF BYTE;
END_TYPE

TYPE
  ALIGN_STRING_20_2 : ARRAY[0..106] OF BYTE;
END_TYPE

TYPE STRING_20_1:
STRUCT
  Token     : TYPE_STRING_20;
  alignment : ALIGN_STRING_20_1;
END_STRUCT;
END_TYPE

TYPE STRING_20_2:
STRUCT
  Token     : TYPE_STRING_20;
  alignment : ALIGN_STRING_20_2;
END_STRUCT;
END_TYPE

TYPE OFFSET:
STRUCT
  X: UINT;
  Y: UINT;
END_STRUCT;
END_TYPE

TYPE
T2_FARBE: ARRAY[0..2] OF UINT;
END_TYPE

TYPE
  F1_2:ARRAY[0..17] OF BYTE;
END_TYPE

TYPE
  T2_TEXT : ARRAY[0..1] OF STRING_20_1;
END_TYPE

TYPE VE_CHANNEL_DATA_CH_1:
STRUCT
FARBE: T2_FARBE;  (* index = 0 *)
    f1 : F1_2;
    TEXT: T2_TEXT;   (* index = 1 *)
END_STRUCT;
END_TYPE

TYPE
  T3_OFFSET : ARRAY[0..1] OF OFFSET;
END_TYPE

TYPE
```

```
  T3_OFFSET : ARRAY[0..1] OF OFFSET;
END_TYPE

TYPE VE_GLOBAL_DATA_FROM_CH_1:
STRUCT
    INFO_IN: T3_OFFSET;
    INFO_OUT: T3_OFFSET;
END_STRUCT;
END_TYPE
```

3.2 Example 2- Use long text strings

3.2.1 V.E. variable list

Assignment in V.E. variable list:

```
#
use_extended_string_var      1    # P-EXTV-00022
#
anzahl_belegt                4
#
var[0].name                  FARBE
var[0].type                  UNS16
var[0].scope                 CHANNEL
var[0].synchronisation       FALSE
var[0].access_rights         READ_WRITE
var[0].array_elements        3
#
var[1].name                  TEXT
var[1].type                  STRING
var[1].scope                 CHANNEL
var[1].synchronisation       FALSE
var[1].access_rights         READ_WRITE
var[1].array_elements        2
#
var[2].name                  INFO_IN
var[2].type                  OFFSET
var[2].scope                 GLOBAL
var[2].synchronisation       FALSE
var[2].access_rights         READ_WRITE
var[2].array_elements        2
#
var[3].name                  INFO_OUT
var[3].type                  OFFSET
var[3].scope                 GLOBAL
var[3].synchronisation       FALSE
var[3].access_rights         READ_WRITE
var[3].array_elements        2
var[3].create_hmi_interface  0
#
struct[0].name               OFFSET
struct[0].element[0].name    X
struct[0].element[0].type    UNS16
struct[0].element[1].name    Y
struct[0].element[1].type    UNS16
#
```

3.2.2 Sample output file for CODESYS

Representation in the exported file:

```
TYPE OFFSET:
STRUCT
    X: UINT;
    Y: UINT;
END_STRUCT
END_TYPE

TYPE VE_CHANNEL_DATA_CH_1:
STRUCT
    FARBE: ARRAY[0..2] OF UINT;
    TEXT: ARRAY[0..1] OF STRING(127);
END_STRUCT
END_TYPE

TYPE VE_GLOBAL_DATA_FROM_CH_1:
STRUCT
    INFO_IN: ARRAY[0..1] OF OFFSET;
    INFO_OUT: ARRAY[0..1] OF OFFSET;
END_STRUCT
END_TYPE
```

3.2.3 PLC example

Integrating the structure in a 3S PLC program:

```
VAR
  (* Use generated structure descriptions *)
  p_ve_chan_1 : POINTER TO VE_CHANNEL_DATA_CH_1;
  p_ve_glob   : POINTER TO VE_GLOBAL_DATA_FROM_CH_1;
  text        : STRING(128);
  init_ve_ptr : BOOL := TRUE;

END_VAR

(* Ensure that the internal management data is initialised *)

Hli(Start := TRUE);


IF Hli.Initialised = TRUE AND Hli.Error = FALSE THEN

  IF init_ve_ptr = TRUE THEN
    (* Provide pointer to structure(s) *)
    p_ve_chan_1 := ADR( gpVECH[0]^.ext_var32[0]);
    p_ve_glob   := ADR(gpVEGlobal^.ext_var32[0]);



  END_IF;

  (* Work with the variables (read, write) *)
  text := p_ve_chan_1^.TEXT[0].token;
  p_ve_chan_1^.FARBE[1] := 2;
END_IF
```

3.3 Example of distance control

3.3.1 V.E. variable list

```
#
use_extended_string_var      0
#
number_used_variables        2
#
var[0].name                  sensor
var[0].type                  SGN32
var[0].scope                 GLOBAL
var[0].synchronisation       FALSE
var[0].access_rights         READ_WRITE
var[0].array_size            0
#
var[1].name                  sensor_ch1
var[1].type                  REAL64
var[1].scope                 CHANNEL
var[1].synchronisation       TRUE
var[1].access_rights         READ_ONLY
var[1].array_size            2
```

3.3.2 Exporting the V.E. variable list

The NC command to export the V.E. variable list for use with CoDeSys is as follows:

```
#EXPORT VE[3S]
```

The default path setting for NC programs is as follows:

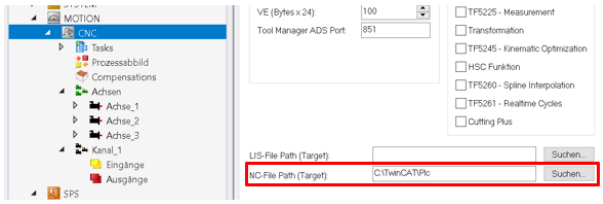

Fig. 1: Example - Default program path setting

The name of the exported file is “plc_3s_ve_types_ch_1.xml” and is in the aboe default path.

3.3.3 Importing the code into PLC

The results of the export can be importing by “right-clicking”.

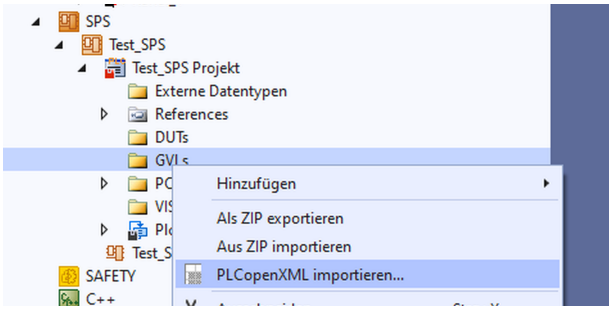

Fig. 2: Importing into development system

After selecting the previously exported XML file, the window below opens:

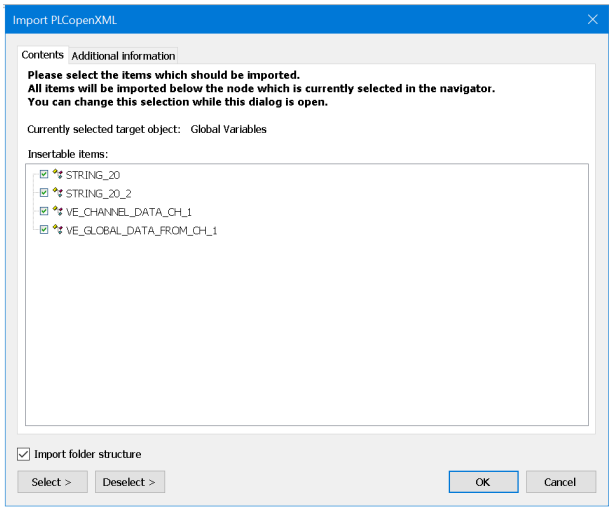

Fig. 3: Import window

```
TYPE VE_CHANNEL_DATA_CH_1 :
STRUCT
  sensor_ch1: ARRAY[0..1] OF LREAL;
END_STRUCT
END_TYPE

TYPE VE_GLOBAL_DATA_FROM_CH_1 :
STRUCT
 sensor: DINT;
END_STRUCT
END_TYPE
```

3.3.4 Integrating into PLC program

```
VAR_GLOBAL
p_ve_global : POINTER TO VE_GLOBAL_DATA_FROM_CH_1;
p_ve_channel: POINTER TO VE_CHANNEL_DATA_CH_1;
END_VAR

(* Ensure that the internal management data is initialised *)
Hli(Start := TRUE);

IF Hli.Initialised = TRUE AND Hli.Error = FALSE THEN

  IF init_ve_ptr = TRUE THEN
    (* Provide pointer to structure(s) *)
    p_ve_chan := ADR(gpVECH[0]^.ext_var32[0]);
    p_ve_glob := ADR(gpVEGlobal^.ext_var32[0]);
  END_IF;

(* Transfer variable sensor values *)
p_ve_global^.sensor := LREAL_TO_DINT(vz_sensor * SENSOR_ENCODER_OUT);

END_IF
```

4 Restrictions in the case of a multi-channel controller structure

When the CNC starts up, the "GLOBALLY" declared variables for each NC channel are added incrementally to any existing variables. The memory layout in its entirety is only defined after start-up has finished. The start address to the common memory is then made available to the PLC.

- The #EXPORT function can only be started in one channel.
- Therefore, it only uses the "GLOBAL" variables declared in that channel. Variables from other channels that are assigned different index values, for example, are invisible. Therefore, they are not entered in the structure VE_GLOBAL_DATA_FROM_CH_<i>.
- A separate VE_GLOBAL_DATA_FROM_CH_<i> structure is created for each channel-specific V.E list where

Recommendation

Identical Global Variables in several channels are created in each of the channels.

5 Short instructions using the export functionality for V.E. variables

Procedure based on TwinCAT

1. Exporting V.E variables from the CNC using the export command #EXPORT VE[TWINCAT]
2. Open the export file with an editor and check for any warnings or errors. These are displayed by a text in the file.
3. Importing the export file to the existing PLC project
4. Create pointers to structures
   (* Use the generated structure descriptions *)
   p_ve_chan_1 : POINTER TO VE_CHANNEL_DATA_CH_1;
   p_ve_glob : POINTER TO VE_GLOBAL_DATA_FROM_CH_1;
5. Assign the addresses of the V.E-specific HLI ranges only once as shown in the example of the defined pointer variables [▶ 10]
6. Integrate read and write access to structures
   p_ve_glob^.VARIABLE_1 := 22; (*Write access*)
   gl_ar_var_3 := p_ve_glob^.VARIABLE_1; (*Read access*)

6 Parameter

P-EXTV-00022	Number of characters of a string variable
Description	This parameter can increase the permissible number of characters of string variables from 21 to 128 characters (each including the termination mark). If the addresses of the V.E. variable is specified in 24-byte blocks (see Memory layout), make sure that 128-byte variables of the STRING type are assigned several 24-byte blocks in the memory layout and that the index is incremented accordingly (cf. variable arrays).
Parameter	use_extended_string_var
Data type	BOOLEAN
Data range	TRUE, FALSE
Dimension	----
Default value	FALSE
Remarks	

7 Appendix

7.1 Suggestions, corrections and the latest documentation

Did you find any errors? Do you have any suggestions or constructive criticism? Then please contact us at documentation@isg-stuttgart.de.

The latest documentation is posted in our Online Help (DE/EN):

QR code link: https://www.isg-stuttgart.de/documentation-kernel/

The link above forwards you to:

https://www.isg-stuttgart.de/fileadmin/kernel/kernel-html/index.html

Notice

Change options for favourite links in your browser;

Technical changes to the website layout concerning folder paths or a change in the HTML framework and therefore the link structure cannot be excluded.

We recommend you to save the above "QR code link" as your primary favourite link.

PDFs for download:

DE:
https://www.isg-stuttgart.de/produkte/softwareprodukte/isg-kernel/dokumente-und-downloads

EN:
https://www.isg-stuttgart.de/en/products/softwareproducts/isg-kernel/documents-and-downloads

Email: documentation@isg-stuttgart.de

Keyword index

P

ISG Industrielle Steuerungstechnik GmbH
STEP, Gropiusplatz 10
D-70563 Stuttgart

www.isg-stuttgart.de
support@isg-stuttgart.de