SYS TEC ELECTRONIC

Dynamic Object Dictionary

Manual

Edition December 2014

This manual includes descriptions for copyrighted products that are not explicitly indicated as such. The absence of the trademark (©) symbol does not infer that a product is not protected. Additionally, registered patents and trademarks are similarly not expressly indicated in this manual.

The information in this document has been carefully checked and is believed to be entirely reliable. However, SYS TEC electronic GmbH assumes no responsibility for any inaccuracies. SYS TEC electronic GmbH neither guarantees nor accepts any liability whatsoever for consequential damages resulting from the use of this manual or its associated product. SYS TEC electronic GmbH reserves the right to alter the information contained herein without prior notification and does not accept responsibility for any damages which might result.

Additionally, SYS TEC electronic GmbH neither guarantees nor assumes any liability for damages arising from the improper usage or improper installation of the hardware or software. SYS TEC electronic GmbH further reserves the right to alter the layout and/or design of the hardware without prior notification and accepts no liability for doing so.

Contact	**Direct**	**Your local distributor**
Address:	SYS TEC electronic GmbH Am Windrad 2 D-08468 Heinsdorfergrund GERMANY	Please find a list of our distributors under http://www.systec-electronic.com/distributors
Ordering Information:	+49 (0) 3765 / 38600-2110 info@systec-electronic.com	
Technical Support:	+49 (0) 3765 / 38600-2140 support@systec-electronic.com	
Fax:	+49 (0) 3765 / 38600-4100	
Web Site:	http://www.systec-electronic.com	

3. Edition December 2014

Table of contents

List of figures and tables

List of abbreviations

DCF	Device Configuration File
OD	Object Dictionary
OpenPCS	Programming tool of the company infoteam for programming PLCs in a programming language according to IEC61131-3
PDO	Process Data Object
PI	Process Image
PLC	Programmable Logic Control
ro	Read only, read permission
rw	Read and write permission
SDO	Service Data Object

1 Introduction

An existing static object dictionary can be extended by PDOs and process variables with module CcmDyn.c. This extension of the object dictionary is called dynamic object dictionary.
The configuration of an object dictionary is another application for the module. Object entries to be complemented and default values of entries are filed in the compact format “Concise configuration storage" according to CiA DS-302.

2 Basics

The object dictionary of a CANopen device consists of a static part and can be complemented by a dynamic part. The static part is defined in the firmware and contains all basic communication objects (SDO, Heartbeat, Emergency, PDOs). The dynamic part of the object dictionary is optional and can be used to add more process variables and PDOs. For a dynamic object dictionary, object entries are generated once during runtime. Afterwards, they can be accessed in the same way as static object entries are accessed.

The DCF file (**D**evice **C**onfiguration **F**ile) for a CANopen node which is generated during a network configuration builds the basis for generating a dynamic object dictionary. The DCF file contains the configured device data and is generated by a CANopen configurator based on an EDS file (**E**lectronic **D**ata **S**heet).

Figure 1 shows the integration of process variables for CANopen for the example of an in OpenPCS programmable PLC. The module CcmDyn.c is used for the CANopen layer for PLCs of the company SYS TEC electronic.

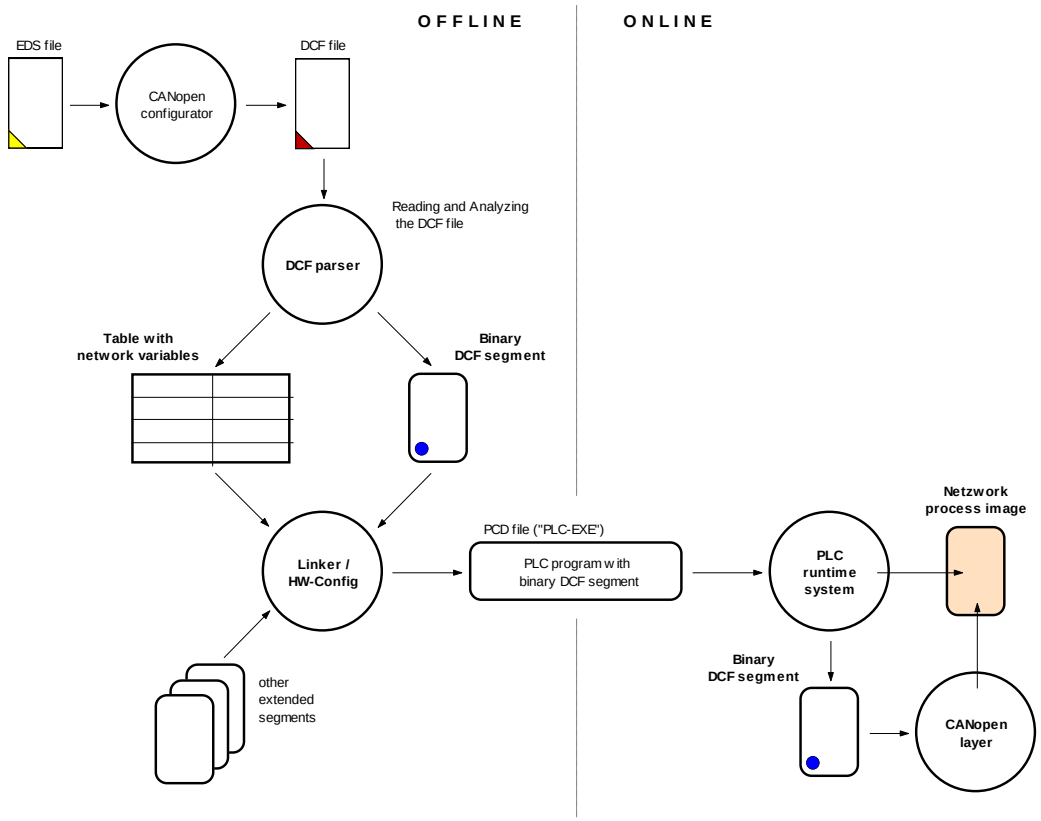

Figure 1: Integration of dynamic process variables for CANopen

PC-sided the "DCF-Parser" (Dcf2Bin.dll) is in charge of reading and evaluating DCF information. Firstly, it reads all relevant sections of the DCF file and sets up an abstract, binary data structure. It serves as basis for dissolving references to network variables and for generating CANopen control information in the form of binary DCF segments. Upon download of the PLC program, the binary DCF segment reaches the control and from there is transferred to the CANopen layer.

For further information like the application programming interface, please refer to the readme.txt file accompanied with the Dcf2Bin.dll.

With the help of the binary DCF segment, new object entries and existing (static) entries can be parameterized. This also includes the reconfiguration of existing PDO objects. *Figure 2* exemplifies this procedure.

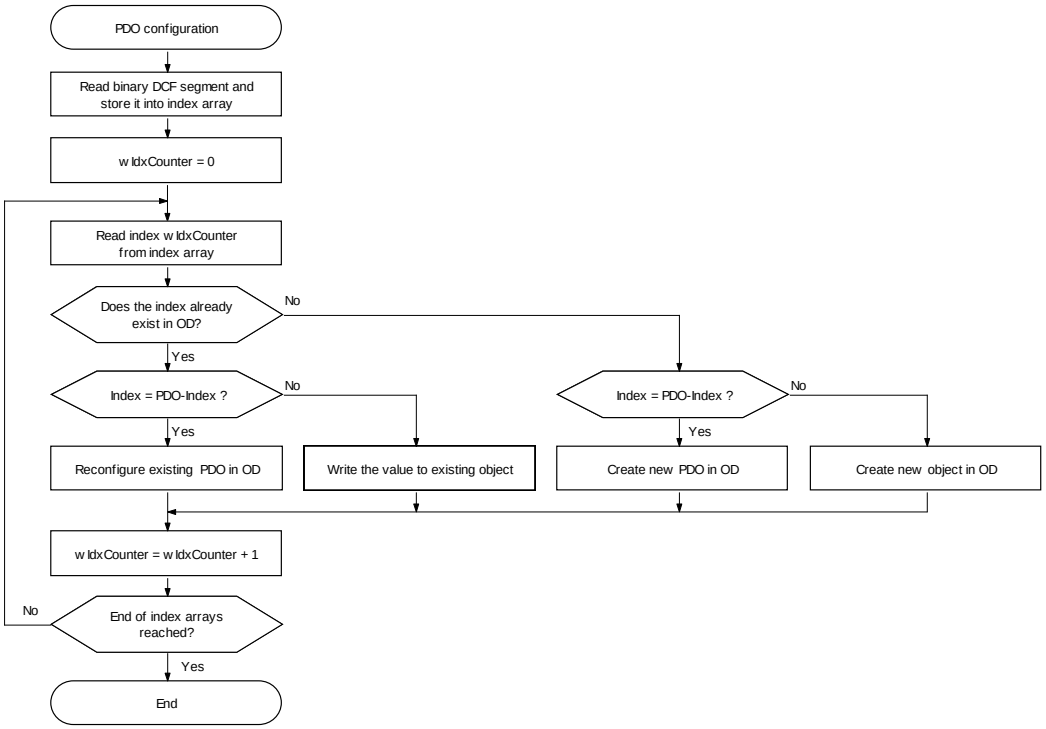

Figure 2: PDO reconfiguration for the generation of the dynamic OD

Advice: PDOs that are reconfigured by the dynamic object dictionary receive their original default values of the static object dictionary when the dynamic OD is destroyed.

2.1 Structure and format of the binary DCF segment

The binary DCF segment forms a container structure for receiving more embedded segments. It contains as subset (index segment) the concise DCF file ("Concise configuration storage") which is a structure defined according to CiA Standard DS-302. *Figure 3* shows the structure of the container for the binary DCF segment.

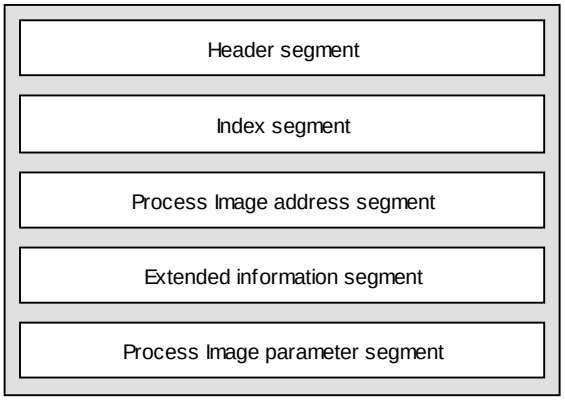

Figure 3: Structure of the container for the binary DCF segment

The binary DCF segment is also used for describing network variables for PLCs. For this task, segments „Process Image address segment“ and „Process Image parameter segment“ are necessary. Those segments are described in the following sections, even though these segments may not apply for the generation of a dynamic object dictionary. The entries in the Header segment then can be set to zero.

2.1.1 Header segment

The Header segment contains the length of each segment and the total length of the Container segment. *Figure 4* describes the segment structure.

DWORD	Size of binary DCF segment
WORD	Version
WORD	Number of segments
DWORD	Offset Index-Segment
DWORD	Size Index-Segment
DWORD	Offset Process Image Adress-Segment
DWORD	Size Process Image Adress-Segment
DWORD	Offset Extended Info-Segment
DWORD	Size Extended Info-Segment
DWORD	Offset Process Image Parameter-Segment
DWORD	Size Process Image Parameter-Segment

Figure 4: Structure of the Header segment

The total length is the sum of all lengths of each single segment (including the length of the Header segment).

All values are entered in the Little-Endian-Format (LSB first). The offset value and the segment length must be set to zero for optional segments that are not used.

2.1.2 Index segment

The structure of the Index segment corresponds to the definition for a shortened DCF file ("Concise configuration storage") according to CiA Standard DS-302. *Figure 5* describes the segment structure.

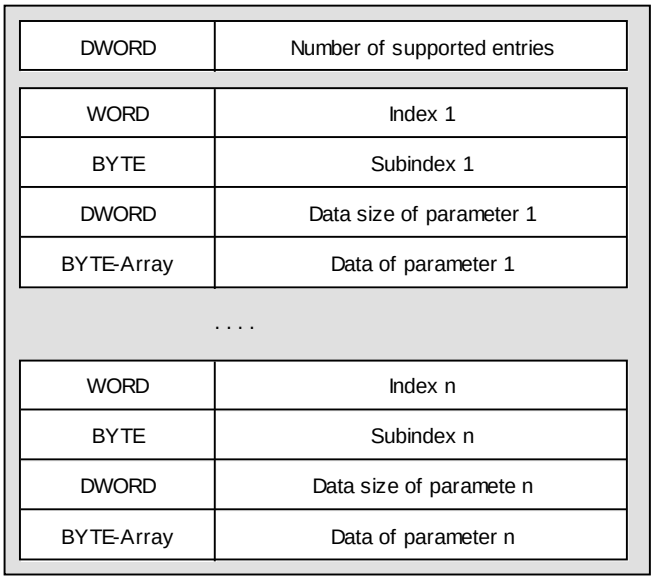

Figure 5: Structure of the Index segment according to CiA DS-302

Advice:
There must be a PDO mapping parameter object for each PDO communication parameter object, e.g. 0x1400 requires object 0x1600 and 0x1801 requires object 0x1A01.
Objects within the address range 0xA000-0xAFFF are always generated as process variables with the following attributes:

NUMERIC_VALUE, READ_PERMISSION, WRITE_PERMISSION, PDO_MAPP_PERMISSION

If the Process image address segment and a process image (network process image) exist, the variables are placed within this memory area.

2.1.3 Process image address segment

The Address segment is an optional part of the binary DCF segment and does not necessarily have to be in the container structure. But the segment is needed when object entries are placed in index range 0xA000-0xAFFF within a network process image.

Background:
The offsets of each variable in the network process image result from their index and subindex. Section 6 describes the algorithm to calculate variable offsets according to CiA Standard DS-302. The variable offset again is provided in the Address segment and forms a redundant data structure.

Upon reading and evaluating the DCF information on the PC side by the "DCF-Parser" (Dcf2Bin.dll), the necessary address calculation takes place by using index and subindex according to the standard. On the PC side, those addresses are used by the linker to solve references to network variables. This also sets the positioning and arrangement of the variables in the network process image. The CANopen layer of the PLC must also use the addresses used for the PLC program (irrespective if those have been calculated norm-compliant or not). If not, there may be inconsistencies between the PLC program/run-time system and the CANopen layer when using the network process image.

To ensure a consistent usage of the network process image between PLC program/run-time system and CANopen layer, PLC-sided the Address segment generated by the "DCF-Parser" (if available) is used for the setup of the dynamic object dictionary. The Address segment is a WORD field which contains an address offset for each variable generated in the network process image.

The Address segment is always available when using the programming system OpenPCS, because the necessary address calculation is needed on the PC side to solve references of network variables. Generally, the functionality that is necessary to set up and administer a dynamic object dictionary can also be used independently from the PLC system. Then the setup of the binary DCF segment takes place by an external component. In this case, it could also be possible that the Address segment is not applicable and the necessary address calculation must take place on the local target system.

2.1.4 Extended info segment

The Extended info segment is a BYTE field which contains the appropriate access attributes for each subindex in the Index segment.

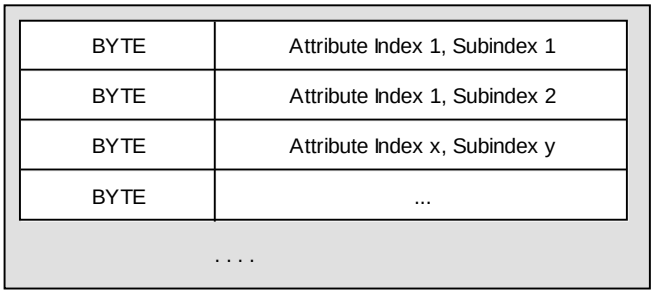

Figure 6: Structure of the Extended info segment

The attributes are defined as bit-masks and can be combined with one another. This could be:

Attribute	Value	Description
BOOLEAN	0x01	The object entry is defined as BOOLEAN.
VISIBLE_STRING	0x08	The object entry is defined as VISIBLE_STRING. The data byte by byte is copied into the object entry.
NUMERIC_VALUE	0x10	The object entry is defined as NUMERIC_VALUE. The data is generated either LSB-first or MSB-first according to the target.
READ_PERMISSION	0x20	Reading the object entry is permitted.
WRITE_PERMISSION	0x40	Writing the object entry is permitted.
PDO_MAPP_PERMISSION	0x80	Mapping the object entry into a PDO is permitted.

Table 1: Access attributes for dynamic object entries

If the attributes BOOLEAN, VISIBLE_STRING or NUMERIC_VALUE are not defined, the object entry is interpreted as DOMAIN. This implies that data is copied byte by byte from the Index segment of the respective object entry.

Examples:

Value	Description
0x70	Numeral value with read and write permission
0x60	Domain with read and write permission
0xF0	Numeral value with read and write permission, mappable into a PDO
0x30	Numeral value with read permission only

2.1.5 Process image parameter segment

This segment contains information about an existing process image (network process image). The segment structure is described in *Figure 7*.

DWORD	Complete size of dyn. Process Image
DWORD	Offset of input area
DWORD	Size of input area
DWORD	Offset of output area
DWORD	Size of output area

Figure 7: Structure of the Process image parameter segment

This segment is not needed to generate dynamic object entries.

3 Interface to the application

To generate a dynamic object dictionary, a binary DCF segment as described in *section 2.1* must be loaded into the Target.[1] This data structure is transferred as parameter to function **DynBuildOd**. The function adds to the existing static object dictionary the dynamic object entries and stores the data in the object entries.

Adding object entries requires a dynamic memory administration. With the help of macros (COP_MALLOC, COP_FREE), platform-specific functions such as *malloc* or *free* can be mounted. Adjustments to the macros take place in file target.h.

Through the dynamic OD it is possible that more PDOs are defined. Existing PDOs in the static object dictionary are set to invalid prior to generating a new PDO. This causes the lost of parameters for the PDO linking. After generating the object dictionary those PDOs again must be linked.

Storing object data of dynamic object entries in a non-volatile memory (e.g. EEPROM) is not supported by the module CcmStore.

For dynamic index entries, callback functions cannot be used in connection with accesses via SDO or with the help of API functions *ObdWriteEntry()* and *ObdReadEntry()* or *CcmWriteObject()* and *CcmReadObject()*.

Creating the dynamic object dictionary should take place after *calling CcmConnectToNet()*. Object entries are generated and PDOs are added.

Recurrent loading of the binary DCF segment into the target and generation of the dynamic object dictionary requires that the existing dynamic object dictionary was logged out and the dynamic allocated memory was released. This task is done by function **DynDestroyOd**.

The following source code extract exemplifies this procedure. The complete source including the binary DCF segment are located in file ex_dynod.c.

[1] Generating and loading the binary DCF segment is subject to the user.

```
// ============================================================
// segment container (contains the build up rules of all
// objects which should be created dynamicaly and/or which should
// be updated with a new value)
// ============================================================
static CONST BYTE ROM abSegmentContainerRom_l[] =
{
    ...
};

// memory for the Process Image (see PI-Param-Segement)
static WORD MEM awProcessImage_l[0x0020];

void main (void)
{
tCopKernel  Ret = kCopSuccessful;
tProcessImageDscrpt PIDscrpt;

    // init the CANopen Stack
    Ret = CcmInitCANopen (&CcmInitParam_g, kCcmFirstInstance);
    ...

    // set CANopen from state INITIALIZATION to PRE-OPERATIONAL
    Ret = CcmConnectToNet ();
    ...

    // fill out the Process Image Descriptor
    PIDscrpt.m_pbProcessImage = (BYTE FAR*) &awProcessImage_l[0];
    PIDscrpt.m_dwProcessImageSize = sizeof (awProcessImage_l);

    // build the dynamic part of the OD
    Ret = DynBuildOd ((BYTE FAR*) &abSegmentContainerRom_l[0],
        &PIDscrpt);
    ...

    while (APP_RUN_FLAG())
    {
        // main prozess function for the CANopen stack
        CcmProcess ();
        ...
    }

    // destroy a previously build dynamic OD
    Ret = DynDestroyOd ();
    ...
}
```

4 Configuring the CANopen Stack

Depending on the application of the dynamic object dictionary, constants must be set in the configuration file *CopCfg.h*:

OBD_USE_DYNAMIC_OD
With the file CcmDyn.c, objects can be dynamically added (during runtime) to the actual object dictionary. To use this feature, this Define must be set to TRUE.

Chosen setting:	TRUE
Range of value:	FALSE, TRUE
Area of application:	CANopen –Source code with dynamic OD

OBD_USER_OD, OBD_USE_VARIABLE_SUBINDEX_TAB
The dynamic object dictionary is processed as USER_OD. To add object entries, variable subindex tables are necessary. Both values must be set to TRUE.

Chosen setting:	TRUE
Range of value:	FALSE, TRUE
Area of application:	CANopen –Source code with dynamic OD

PDO_USE_BIT_MAPPING
To add and map object entries of type BOOLEAN, the value must be set to TRUE. Otherwise, mapping of bit values is not possible.

Chosen setting:	TRUE
Range of value:	FALSE, TRUE
Area of application:	CANopen –Source code

PDO_GRANULARITY
To map object entries of type BOOLEAN at any place in the PDO, this value must be set to 64. Otherwise, a maximum of 8 bit is mappable.

Chosen setting:	64
Range of value:	8, 64
Area of application:	CANopen –Source code

5 Description of the application functions

5.1 Function DynBuildOd

Syntax:

```
#include <CcmDyn.h>
tCopKernel PUBLIC DynBuildOd ( CCM_DECL_INSTANCE_HDL_
             BYTE FAR* pbDcfSegCont_p,
             tProcessImageDscrpt* pProcessImageDscrpt_p);
```

Parameter:

CCM_DEC_INSTANCE_HDL_: Instance-Handle

pbDcfSegCont_p:	Pointer to the binary DCF segment
pProcessImageDscrpt_p:	Pointer to a data structure to describe the process image

Return:

kCopSuccessful	0x00	The function was executed without error.
kCopIllegalInstance	0x01	The parameterized instance does not exist.
kCopCobNoFreeEntry	0x20	No free entry was found in the COB table. The COB could not be created.
kCopCobAllreadyExist	0x21	No COB can be registered for the parameterized CAN identifier because this CAN identifier is already used for another COB.

kCopCobIllegalCanId	0x23	The CAN identifier does not correspond to the value range for an identifier and is therefore not accepted (CAN-ID = 0 and the 29Bit identifier is not supported by the current configuration of the CANopen stack).
kCopObdIndexNotExist	0x31	The object index is not defined in the current OD.
kCopObdSubindexNotExist	0x32	The object entry is not found under the parameterized sub-index.
kCopPdoErrorMapp	0x78	When mapping variables to a PDO, an error was discovered as a result of invalid object entry parameters.
kCopDynNoMemory	0xA0	Not enough memory to build up the dynamic OD.
kCopDynInvalidConfig	0xA1	Invalid configuration in the Segment Container for assembly of the OD

Description:

The function complements dynamic object entries from a binary DCF segment. Existing object entries and data are loaded from the binary DCF segment if those entries are available in the DCF segment.
If the function is used to generate complemented dynamic object entries within a process image (network process image), the start address (m_pbProcessImage) and the maximum available size of the process image (m_dwProcessImageSize) must be transferred by using structure *tProcessImageDscrpt*.

```
typedef struct
{
    // Address and size of process image
    BYTE FAR* m_pbProcessImage;     // IN-Parameter
    DWORD     m_dwProcessImageSize; // IN/OUT-Parameter

    // Offset and size of inputs
    DWORD     m_dwDynPIOffsetIn;    // OUT-Parameter
    DWORD     m_dwDynPISizeIn;      // OUT-Parameter

    // Offset and size of outputs
    DWORD     m_dwDynPIOffsetOut;   // OUT-Parameter
    DWORD     m_dwDynPISizeOut;     // OUT-Parameter

} tProcessImageDscrpt;
```

If the segment ProcessImage-Param is available in the binary DCF segment, the function provides the contained parameters in structure *tProcessImageDscrpt*.

5.2 Function DynDestroyOd

Syntax:

```
#include <CcmDyn.h>
tCopKernel PUBLIC DynDestroyOd (  CCM_DECL_INSTANCE_HDL);
```

Parameter:

CCM_DEC_INSTANCE_HDL: Instance-Handle

Return:

kCopSuccessful	0x00	The function was executed without error.
kCopIllegalInstance	0x01	The parameterized instance does not exist.

Description:

The function deletes complemented dynamic object entries and resets the PDO module.
Before the dynamic object dictionary can be deleted, the communication objects for the PDOs must be reset. Therefore, the NMT-Event-Function of the PDO module must be called with event kNmtEvPreResetCommunication.
Afterwards, the dynamic object dictionary and the generated PDO tables are deleted and the default values for the communication parameters of the PDOs are set in the static object dictionary.
Then, the PDO module is actuated to state PreOperational.

6 Index entries and allocating variable offsets

The CiA Standard DS-405 defines the index range for network variables listed in *Table 2* to be used by the IEC 61131-3. Through the programming system OpenPCS, only parts of the possible variable types are supported.

According to the CiA Standard DS-302, the allocation of offsets for network variables is done by the CANopen configurator. For each variable type (data type and access direction), a separate subsegment is generated. The positioning and size of these subsegments (PIOffset=, MaxCnt=, Range=) within the network process image can be optionally defined by the EDS file. If the EDS file does not contain such entries, the start offset for a subsegment is considered as zero and the size of the subsegment adjusts dynamically to the amount of declared variables. Process images are generated separately for input and output variables.

Each subsegment must be interpreted as an array of the respective variable type (BYTE, WORD, DWORD, ...). The subindex for which a variable was generated in the DCF file, reflects its array index. Hereby must be considered that the first subindex for a variable definition holds value 1 while in high level language (e.g. C) the first array element holds index 0. To get the actual array index of a variable in the process image, the respective subindex of the variable must be decremented by value 1. The subindex indicates the array element which is allocated to the variable and does not indicate the absolute offset within the process image. For example, with subindex=6 for a byte variable, the offset 5 in the process image is addressed (=(6-1)*sizeof(BYTE)), but with Subindex=3 of a DWORD variable, offset 8 (=(3-1)*sizeof(DWORD)).

Data direction	Start index	Data type	Access type	Usage in OpenPCS
Input	A000H	Integer8	ro	x
	A040H	Unsigned8	ro	x
	A080H	Boolean	ro	-
	A0C0H	Integer16	ro	x
	A100H	Unsigned16	ro	x
	A140H	Integer24	ro	-
	A180H	Unsigned24	ro	-
	A1C0H	Integer32	ro	x
	A200H	Unsigned32	ro	x
	A240H	Float (32)	ro	(x)
	A280H	Unsigned40	ro	-
	A2C0H	Integer40	ro	-
	A300H	Unsigned48	ro	-
	A340H	Integer48	ro	-
	A380H	Unsigned56	ro	-
	A3C0H	Integer56	ro	-
	A400H	Integer64	ro	-
	A440H	Unsigned64	ro	-
Output	A480H	Integer8	rw	x
	A4C0H	Unsigned8	rw	x
	A500H	Boolean	rw	-
	A540H	Integer16	rw	x
	A580H	Unsigned16	rw	x
	A5C0H	Integer24	rw	-
	A600H	Unsigned24	rw	-
	A640H	Integer32	rw	x
	A680H	Unsigned32	rw	x
	A6C0H	Float (32)	rw	(x)
	A700H	Unsigned40	rw	-
	A740H	Integer40	rw	-
	A780H	Unsigned48	rw	-
	A7C0H	Integer48	rw	-
	A800H	Unsigned56	rw	-
	A840H	Integer56	rw	-
	A880H	Integer64	rw	-
	A8C0H	Unsigned64	rw	-

Table 2: Index range for network variables

Each index entry can manage an array with up to 254 elements for the respective data type (BYTE, WORD, DWORD, ...). To generate more variables, the following index entry must be used. For example, index 0A4C0H can manage the first 254 byte variables and from the 255th byte variable the index 0A4C1H is necessary.

For the allocation of subindexes, the CANopen configurator takes into consideration possible overlapping of subsegments. The variable offset can be chosen so that each variable is allocated a separate memory area according to its type. Overlapping subsegments exist if the EDS file does not contain presetting in the form of start offsets for each variable area. In this case, all subsegments by default start from offset zero.

The CANopen configurator places all variables of the first subsegment at the beginning of the process image (if not all subsegments have been displaced by the EDS file to start offsets above zero) and connects the variables of the following subsegments at it. The subindex is an indirect equivalent for the offset of the variables in the process image. Consequently, the subindexes of the variables in the first subsegment start with value 1. The subindex of the variable of all following subsegments (variable types) depends on the memory already taken and the first available array index for the respective variable type within the process image *(see Figure 8)*.

The following extract from a DCF file in combination with *Figure 8* exemplifies the connection between index/subindex and offset in the process image.

```
[A4C0]
SubNumber=4

[A4C0sub0]
ParameterName=NrOfElements

[A4C0sub1]
ParameterName=IN0_IN7

[A4C0sub2]
ParameterName=IN8_IN15

[A4C0sub3]
ParameterName=IN16_IN23

[A580]
SubNumber=2

[A580sub0]
ParameterName=NrOfElements

[A580sub3]
ParameterName=AIN0
```

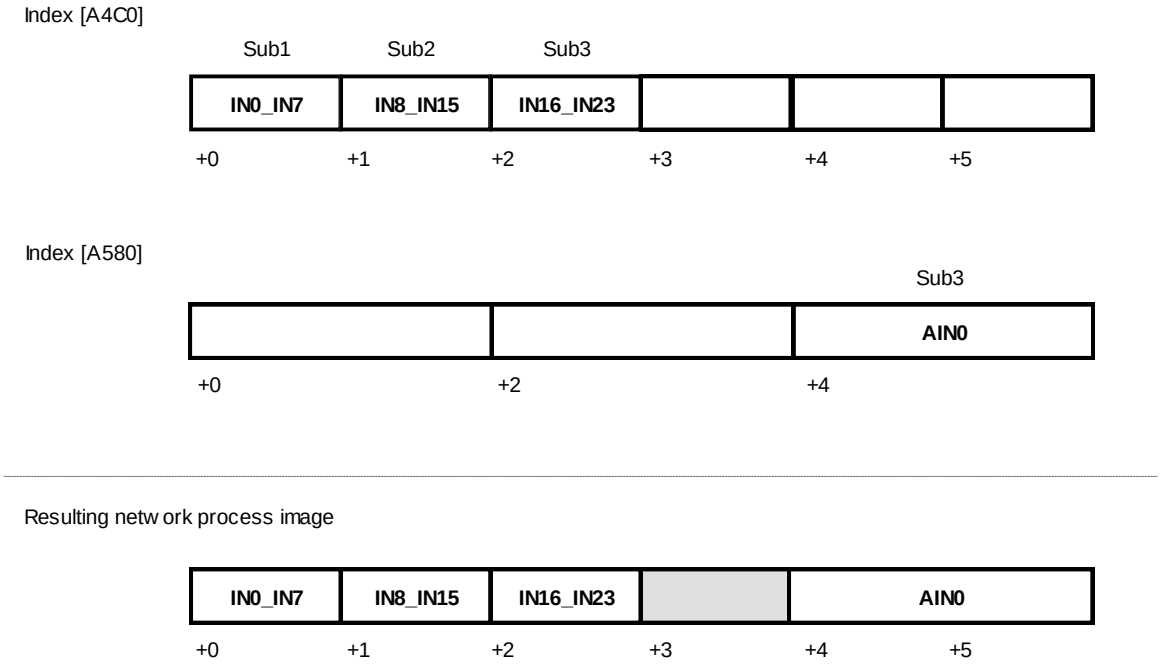

Figure 8: Example for the connection between index/subindex and variable offset

Index

Document:	Dynamic Object Dictionary
Document number:	L-1087e_3, Edition December 2014

How would you improve this manual?

Have you spot any mistake in this manual? Page

Submitted by:

Customer number: _____

Name: _____

Company: _____

Address: _____

Please send to: SYS TEC electronic GmbH
August-Bebel-Str. 29
D-07973 Greiz
GERMANY
Fax : +49 (0) 36 61 / 62 79 99

Published by

SYS TEC ELECTRONIC

Best.-Nr. L-1087e_3
Printed in Germany