HISTORY

Date                  Changes
March 1998            Document Version 1.0 released
December 2000         Document Version 2.0 released
                       • editorial changes
                       • adaptation to Version 4.01 of CiA DS-301
                       • adaptation to Version 3.0 of CiA DSP-302
                       • function blocks for SDO with arbitrary length
                       • Tool Integration redefined
                       • clarifications
May 2002              Document as Draft Standard Version 2.0 released

General information on licensing and patents

CAN in AUTOMATION (CiA) calls attention to the possibility that some of the elements of this CiA specification may be subject of patent rights. CiA shall not be responsible for identifying any or all such patent rights.

Because this specification is licensed free of charge, there is no warranty for this specification, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holder and/or other parties provide this specification “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the correctness and completeness of the specification is with you. Should this specification prove failures, you assume the cost of all necessary servicing, repair or correction.
TABLE OF CONTENTS

1 SCOPE 1

2 REFERENCES 2

3 TERMS AND DEFINITIONS 2
  3.1 DATA TYPES 2
  3.2 VARIABLES 3
  3.3 PROGRAM ORGANIZATION UNITS (POU) 3
  3.4 CONFIGURATION ELEMENTS 4
  3.5 ACCESS TO NETWORK VARIABLES DEFINITION 4
  3.6 CANOPEN PROFILE-SPECIFIC DEFINITIONS 4

4 DATA TYPES 5
  4.1 DATA TYPE CONVERSION FUNCTIONS 6
  4.2 DATA TYPE REPRESENTATION ISSUES 7

5 ACCESSING CANOPEN FROM WITHIN IEC 61131-3 8
  5.1 VARIABLE BASED ACCESS 8
    5.1.1 Definition of the data direction 8
    5.1.2 Object dictionary entries for IEC 61131-3 variables 9
    5.1.3 Variable names 10
  5.2 FUNCTION BLOCK BASED ACCESS 11
    5.2.1 General function block design issues 12
    5.2.2 Data types 12
    5.2.3 SDO access 14
    5.2.4 Other function blocks 17

6 TOOL INTEGRATION 20
  6.1 BASIC CONCEPT 20
  6.2 NODE LIST 20
  6.3 ACCESS HANDLER 21

7 UTILITY FUNCTIONS 22
  7.1 REMOTE FUNCTIONS BETWEEN RESOURCES 22
    7.1.1 Down-/Upload 22
  7.2 OBJECT DICTIONARY ENTRIES 22
    7.2.1 Project name 22
    7.2.2 Configuration 23
    7.2.3 Resources 24
    7.2.4 Task 25
    7.2.5 Start/Stop (Program/Task) 27
    7.2.6 Debugging/Monitoring 27

8 APPENDIX (INFORMATIVE) 28
  8.1 DETAILED DESCRIPTION OF NETWORK VARIABLE SEGMENTS 28
  8.2 IEC 61131-3 OBJECT DICTIONARY OVERVIEW 29
  8.3 EXAMPLE DCF FILE 31
  8.4 APPLICATION NOTES 31
    8.4.1 Network variables 31
    8.4.2 Data type representation issues 32
    8.4.3 EDS 33
    8.4.4 Tool integration 33
  8.5 IEC 61131-3 SAMPLE CODE 35
  8.6 IMPLEMENTATION MODELS FOR IEC 61131-3 DATA TYPE SUPPORT 35
    8.6.1 Native support 35
    8.6.2 Padding to next best match 35
    8.6.3 Using arrays or struct 36
FIGURES

Figure 1: Interfaces ................................................................................................................1
Figure 2: IEC 61131-3 software organisation.........................................................................3
Figure 3: Direction of Network Variables ...............................................................................8
Figure 4: Type CIA405_DEVICE ............................................................................................12
Figure 5: Type CIA405_SDO_ERROR ...............................................................................12
Figure 6: Type CIA405_EMCY_ERROR .............................................................................12
Figure 7: Type CIA405_STATE .............................................................................................13
Figure 8: Type CIA405_TRANSITION_STATE ...................................................................13
Figure 9: Type CIA405_CANOPEN_KERNEL_ERROR .........................................................13
Figure 10: Function block CIA405_SDO_WRITE4 .........................................................14
Figure 11: CIA405_SDO_WRITE4 typical timing diagram ..................................................14
Figure 12: Function block CIA405_SDO_WRITE .............................................................15
Figure 13: Function block CIA405_SDO_READ .................................................................16
Figure 14: CIA405_SDO_READ4 typical timing diagram ..................................................16
Figure 15: Function block CIA405_SDO_READ .................................................................17
Figure 16: Function block CIA405__GET_LOCAL_NODE_ID ........................................17
Figure 17: Function block CIA405__GET_STATE ............................................................18
Figure 18: Function block CIA405__GET_CANOPEN_KERNEL_STATE ........................18
Figure 19: Function block CIA405_NMT ...........................................................................18
Figure 20: Function block CIA405_RECV_EMCY ............................................................19
Figure 21: Function block CIA405_RECV_EMCY ............................................................19
Figure 22: Correlation between programming and network configuration .......................20
Figure 23: Project structure .................................................................................................22
Figure 24: Data representation with PDO on CANopen non-compliant PLCs ..................32
Figure 25: Data representation with SDO on CANopen non-compliant and compliant PLCs 32
Figure 26: Information exchange via DCF ........................................................................33
Figure 27: Data exchange using converter module ..............................................................33
Figure 28: Compatibility for NVX format ............................................................................34

TABLES

Table 1: Data types .................................................................................................................5
Table 2: Data types not transferable via PDO .......................................................................6
Table 3: Data type conversion functions .............................................................................6
Table 4: Input network variables ..........................................................................................9
Table 5: Output network variables ......................................................................................10
Table 6: New data types .......................................................................................................11
Table 7: Function blocks ......................................................................................................11
Table 8: Error codes for type CIA405_CANOPEN_KERNEL_ERROR ................................14
Table 9: Entries in Nodelist file ........................................................................................21
Table 10: List of object dictionary entries ..........................................................................30
1 Scope

This document represents the standardised CANopen Interface and Device Profile for IEC 61131-3 programmable devices like PLCs.

All the above devices use communication techniques which conform to those described in the CiA Draft Standard DS-301 (Application Layer and Communication Profile) and Draft Standard Proposal DSP-302. These documents should be consulted in parallel to this profile.

In general, generating an application implements the handling of up to five interfaces see the figure.

This paper covers the following interfaces

A) the access to a CANopen communication system from within an IEC 61131-3 program
   a) based on variables, i.e. access to elementary IEC 61131-3 variable objects,
   b) based on calls to function block
B) utility functions for debugging, monitoring and network management
C) interface between CANopen tools and IEC 61131-3 programming environment
   All other interfaces are manufacturer specific (D) or uses CANopen services (E).

Note Figure 1 describes not necessarily the use of different tool for programming and configuration. One tool may handle both functionality and hide the interfaces.
2 References

/IEC 61131/ IEC 61131 First edition 1993-03
Programmable controllers
- Part 1: general information
- Part 3: programming languages
- Part 5: messaging services
/DS-301/ Application Layer and Communication Profile
CiA Draft Standard 301 Version 4.01, 2000-06
/DSP-302/ Framework for programmable CANopen devices
CiA Draft Standard Proposal 302 Version 3.0, 2000-06
/DSP-305/ Layer Setting Services and Protocols
CiA Draft Standard Proposal 305 Version 1.1, 2002-02
/DSP-306/ Electronic Data Sheet Specification for CANopen
CiA Draft Standard Proposal 306 Version 1.1, 2001-06

3 Terms and definitions

3.1 Data types
IEC 61131-3 specifies three kinds of data types:

- **Elementary data types**

<table>
<thead>
<tr>
<th>Length</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Bit</td>
<td>BOOL</td>
</tr>
<tr>
<td>8 Bits</td>
<td>SINT, USINT, BYTE</td>
</tr>
<tr>
<td>16 Bits</td>
<td>INT, UINT, WORD</td>
</tr>
<tr>
<td>32 Bits</td>
<td>DINT, UDINT, REAL, DWORD</td>
</tr>
<tr>
<td>64 Bits</td>
<td>LINT, ULINT, LREAL, LWORD</td>
</tr>
<tr>
<td>Implementation dependent</td>
<td>TIME, DATE, TIME_OF_DAY, DATE_AND_TIME, STRING</td>
</tr>
</tbody>
</table>

- **Generic data types**

  Generic data types group elementary data types hierarchically. They are identified by the prefix "ANY" (e.g. ANY_NUM, ANY_INT,...).

- **Derived data types**

  User- or manufacturer-specified data types (Structures). Types to match CANopen data types (e.g. CIA405UINT24, ...).
3.2 Variables

IEC-Variables can be represented symbolically or directly (%IX..., %QW ..). IEC 61131-3 knows two kinds of variables:

- **Single-element variable**
  Variable consists of one element.

- **Multi-element variable**
  Arrays and structures.

3.3 Program organization units (POU)

The POUs are the actual programming modules, an IEC program consists of. The program organisation units defined by IEC 61131-3 include function, function block, and program.

![Diagram of IEC 61131-3 software organisation](image)

*Figure 2: IEC 61131-3 software organisation*

- **Function**
  A function has one or more input values and exactly one output value. It does contain no internal state information, i.e. invocation of a function with the same arguments (inputs) always yields the same value (output). A function can be executed from a program, a function block or another function.

- **Function block**
  A function block has one or more input values and one or more output values. Multiple instances (copies) of a function block are allowed. A function block keeps its internal state information. It can be executed from a program or another function block.
• **Program**
  This POU type represents the "main program". Instances of programs can only be created within resources (by a task).

### 3.4 Configuration elements

The configuration corresponds to a programmable controller system as defined in IEC 61131-3-1. A configuration contains one or more resources, each of which contains one or more programs executed under the control of zero or more tasks.

• **Global variable**
  A variable whose scope is global (Scope of a declaration applying to all POUs within a resource or configuration).

• **Resource**
  "Signal processing function" and its "man-machine interface" and "sensor and actuator interface functions ", as defined in IEC 61131-3-1. Normally a resource means one PLC’s central processing unit.

• **Task**
  A task is defined as an execution control element which is capable of invoking, either on a periodic basis or upon the occurrence of the rising edge of a specific Boolean variable, the execution of a set of program organization units, which can include programs and function blocks.

• **Access path**
  The association of a symbolic name with a variable for the purpose of open communication.

### 3.5 Access to network variables definition

In a network programmable nodes can be characterized as a process having input variables and output variables. The set of variables will be arguments of the program and hence will be only known in a final state when the program has been written. The arguments must be handled as variables located in the object dictionary.

This profile uses the terms and definitions of /DSP-302/ "Framework for programmable CANopen Devices". It defines the usage of network variables in a way independent of the type of the programmable device. Here, some restrictions are made for IEC 61131-3 programmable devices.

### 3.6 CANopen profile-specific definitions

CANopen provides the ability to identify the profile of a node. Therefore the CANopen Communication Profile /DS-301/ specifies an object 1000h ("Device Type"). The object 1000h is a 32 Bit word, subdivided into two 16 Bit words. The LSB contains the profile number, the MSB contains additional information. For a device, following these specification the profile number is set to 405. The additional information (MSB) is set to 0 and is reserved for further use by CiA. The profile does not define a default mapping.
4 Data types

IEC 61131-3 and CANopen define different data types. The following table defines the assignment of types that are equivalent and that are usable as network variables. For a handling of these data types in a file format it is useful to assign numbers to the data type. The table starts with the data types that are defined in CANopen and uses the same coding. Additional data types of IEC 61131-3 start with A0H.

<table>
<thead>
<tr>
<th>IEC 61131-3</th>
<th>CANopen</th>
<th>Data width (bit)</th>
<th>Encoding (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Boolean</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>SINT</td>
<td>Integer8</td>
<td>8</td>
<td>02</td>
</tr>
<tr>
<td>INT</td>
<td>Integer16</td>
<td>16</td>
<td>03</td>
</tr>
<tr>
<td>CIA405INT24</td>
<td>Integer24</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>DINT</td>
<td>Integer32</td>
<td>32</td>
<td>04</td>
</tr>
<tr>
<td>CIA405INT40</td>
<td>Integer40</td>
<td>40</td>
<td>12</td>
</tr>
<tr>
<td>CIA405INT48</td>
<td>Integer48</td>
<td>48</td>
<td>13</td>
</tr>
<tr>
<td>CIA405INT56</td>
<td>Integer56</td>
<td>56</td>
<td>14</td>
</tr>
<tr>
<td>LINT</td>
<td>Integer64</td>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>USINT</td>
<td>Unsigned8</td>
<td>8</td>
<td>05</td>
</tr>
<tr>
<td>UINT</td>
<td>Unsigned16</td>
<td>16</td>
<td>06</td>
</tr>
<tr>
<td>CIA405UINT24</td>
<td>Unsigned24</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>UDWORD</td>
<td>Unsigned32</td>
<td>32</td>
<td>07</td>
</tr>
<tr>
<td>CIA405UINT40</td>
<td>Unsigned40</td>
<td>40</td>
<td>18</td>
</tr>
<tr>
<td>CIA405UINT48</td>
<td>Unsigned48</td>
<td>48</td>
<td>19</td>
</tr>
<tr>
<td>CIA405UINT56</td>
<td>Unsigned56</td>
<td>56</td>
<td>1A</td>
</tr>
<tr>
<td>ULINT</td>
<td>Unsigned64</td>
<td>64</td>
<td>1B</td>
</tr>
<tr>
<td>REAL</td>
<td>Float</td>
<td>32</td>
<td>08</td>
</tr>
<tr>
<td>BYTE</td>
<td>Unsigned8</td>
<td>8</td>
<td>A4</td>
</tr>
<tr>
<td>WORD</td>
<td>Unsigned16</td>
<td>16</td>
<td>A5</td>
</tr>
<tr>
<td>DWORD</td>
<td>Unsigned32</td>
<td>32</td>
<td>A6</td>
</tr>
<tr>
<td>LWORD</td>
<td>Unsigned64</td>
<td>64</td>
<td>A7</td>
</tr>
</tbody>
</table>

Table 1: Data types
The following data types are equivalent, but cannot be transferred via PDOs:

<table>
<thead>
<tr>
<th>IEC 61131-3</th>
<th>CANopen</th>
<th>Encoding (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>Time_Difference</td>
<td>0D</td>
</tr>
<tr>
<td>DATE</td>
<td>Time_Of_Day</td>
<td>0C</td>
</tr>
<tr>
<td>TIME_OF_DAY</td>
<td>Time_Of_Day</td>
<td>0C</td>
</tr>
<tr>
<td>STRING</td>
<td>Visible String</td>
<td>09</td>
</tr>
</tbody>
</table>

Table 2: Data types not transferable via PDO

The following IEC 61131-3 data types do have no equivalent in CANopen: LREAL, DATE_AND_TIME. The following CANopen data types do have no equivalent in IEC 61131-3: Octet String, Domain.

4.1 Data type conversion functions

Type conversion functions shall be defined as in IEC 61131-3 (i.e. `<from>_TO_<to>`, e.g. CIA405uint24_TO_DINT) in both directions for the following combinations:

<table>
<thead>
<tr>
<th>Type1</th>
<th>Type2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIA405INT24</td>
<td>DINT</td>
</tr>
<tr>
<td>CIA405INT40</td>
<td>LINT</td>
</tr>
<tr>
<td>CIA405INT48</td>
<td>LINT</td>
</tr>
<tr>
<td>CIA405INT56</td>
<td>LINT</td>
</tr>
<tr>
<td>CIA405UINT24</td>
<td>UDINT</td>
</tr>
<tr>
<td>CIA405UINT40</td>
<td>ULINT</td>
</tr>
<tr>
<td>CIA405UINT48</td>
<td>ULINT</td>
</tr>
<tr>
<td>CIA405UINT56</td>
<td>ULINT</td>
</tr>
<tr>
<td>CIA405UINT24</td>
<td>DWORD</td>
</tr>
<tr>
<td>CIA405UINT40</td>
<td>LWORD</td>
</tr>
<tr>
<td>CIA405UINT48</td>
<td>LWORD</td>
</tr>
<tr>
<td>CIA405UINT56</td>
<td>LWORD</td>
</tr>
</tbody>
</table>

Table 3: Data type conversion functions
4.2 Data type representation issues

CANopen defines a certain representation for data types, while IEC 61131-3 does not. If the IEC 61131-3 system can use the same representation as defined in CANopen, there will be no problem. The strategy of this standard is as follows:

a) Data of the network (typically PDOs) made available to the IEC 61131-3 as network variables through entries in the range A000-AFFF, typically mapped into the process image of the IEC 61131-3 system shall be presented to the IEC 61131-3 in the IEC 61131-3 system's native (platform dependent, potentially CANopen non-compliant) data representation. The CANopen data type of these data items is known to the CANopen kernel, all necessary conversions will have to be done before putting the value into the process image resp. after retrieving it from the process image.

b) All other data, for which the data type is potentially unknown to the CANopen kernel and/or the IEC 61131-3 application (e.g. SDO data) shall be presented to the IEC 61131-3 system in standard CANopen data format as a stream of raw bytes. Conversion functions shall be supplied to convert this data representation to native IEC 61131-3 format, but the knowledge about the data type has to be available at run-time.

Refer Chapter 8.4.2 for application notes.
5 Accessing CANopen from within IEC 61131-3

As mentioned in the introduction, this paper covers two methods of accessing the CANopen network:

1. Variable-based access, accessing individual variables from within the IEC 61131-3 program according to the Network Variables of /DSP-302/.

2. Function block based access, calling function blocks to write and/or read data. This is implemented with SDO access.

3. Additional management functions for processing Emergency, LSS and Network state information.

5.1 Variable based access

All objects located in the range A000h - AFFFh of the object dictionary of a node shall be visible as variables to an application on that node programmed within IEC 61131-3. Which kind of variables are being used, e.g.

- directly represented variables
- external variables
- parameters (of programs or function blocks)

shall be left to the IEC 61131-3 implementation.

5.1.1 Definition of the data direction

The terms input and output are defined in the Framework for Programmable Devices /DSP-302/. An input has the CANopen data direction ro, an output has the CANopen data direction wo (rww respectively) as with every other device type, too.

Hint:

This definition comes from the network point of view. This shall be explained in Figure 3 by an example for transferring a physical input:

![Diagram](image.png)

Figure 3: Direction of Network Variables

The object A4C0H is defined as an output, since another device can write values to it, which is (from the CANopen point of view) the same as writing a value to a digital output. If the device has a
physical output or a software, that deals with these values - is in principle no difference. So one can use the terminology of “writing values to the PLC application”.

From the PLC programmer’s point of view, the application will read the value – this is the same as if it would read the digital input lines directly without having a bus between.

5.1.2 Object dictionary entries for IEC 61131-3 variables
The Framework for Programmable Devices /DSP-302/ defines the usage of so-called segments. It leaves the concrete placement of the segments open. To ease implementations and for a much easier usage of software from different manufacturers, the usage of the segments is specified for IEC 61131-3 programmed devices: The segments are placed in the index range A000h-AFFFh. This allows any device of another profile (e.g. I/O device or drive unit) to use the standard object dictionary entries of that profile and additionally use the Network Variables.

<table>
<thead>
<tr>
<th>Start-Index</th>
<th>Data Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A000H</td>
<td>Integer8</td>
<td>ro</td>
</tr>
<tr>
<td>A040H</td>
<td>Unsigned8</td>
<td>ro</td>
</tr>
<tr>
<td>A080H</td>
<td>Boolean</td>
<td>ro</td>
</tr>
<tr>
<td>A0C0H</td>
<td>Integer16</td>
<td>ro</td>
</tr>
<tr>
<td>A100H</td>
<td>Unsigned16</td>
<td>ro</td>
</tr>
<tr>
<td>A140H</td>
<td>Integer24</td>
<td>ro</td>
</tr>
<tr>
<td>A180H</td>
<td>Unsigned24</td>
<td>ro</td>
</tr>
<tr>
<td>A1C0H</td>
<td>Integer32</td>
<td>ro</td>
</tr>
<tr>
<td>A200H</td>
<td>Unsigned32</td>
<td>ro</td>
</tr>
<tr>
<td>A240H</td>
<td>Float (32)</td>
<td>ro</td>
</tr>
<tr>
<td>A280H</td>
<td>Unsigned40</td>
<td>ro</td>
</tr>
<tr>
<td>A2C0H</td>
<td>Integer40</td>
<td>ro</td>
</tr>
<tr>
<td>A300H</td>
<td>Unsigned48</td>
<td>ro</td>
</tr>
<tr>
<td>A340H</td>
<td>Integer48</td>
<td>ro</td>
</tr>
<tr>
<td>A380H</td>
<td>Unsigned56</td>
<td>ro</td>
</tr>
<tr>
<td>A3C0H</td>
<td>Integer56</td>
<td>ro</td>
</tr>
<tr>
<td>A400H</td>
<td>Integer64</td>
<td>ro</td>
</tr>
<tr>
<td>A440H</td>
<td>Unsigned64</td>
<td>ro</td>
</tr>
</tbody>
</table>

Table 4: Input network variables
## Outputs:

<table>
<thead>
<tr>
<th>Start-Index</th>
<th>Data Type</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A480H</td>
<td>Integer8</td>
<td>rww</td>
</tr>
<tr>
<td>A4C0H</td>
<td>Unsigned8</td>
<td>rww</td>
</tr>
<tr>
<td>A500H</td>
<td>Boolean</td>
<td>rww</td>
</tr>
<tr>
<td>A540H</td>
<td>Integer16</td>
<td>rww</td>
</tr>
<tr>
<td>A580H</td>
<td>Unsigned16</td>
<td>rww</td>
</tr>
<tr>
<td>A5C0H</td>
<td>Integer24</td>
<td>rww</td>
</tr>
<tr>
<td>A600H</td>
<td>Unsigned24</td>
<td>rww</td>
</tr>
<tr>
<td>A640H</td>
<td>Integer32</td>
<td>rww</td>
</tr>
<tr>
<td>A680H</td>
<td>Unsigned32</td>
<td>rww</td>
</tr>
<tr>
<td>A6C0H</td>
<td>Float (32)</td>
<td>rww</td>
</tr>
<tr>
<td>A700H</td>
<td>Unsigned40</td>
<td>rww</td>
</tr>
<tr>
<td>A740H</td>
<td>Integer40</td>
<td>rww</td>
</tr>
<tr>
<td>A780H</td>
<td>Unsigned48</td>
<td>rww</td>
</tr>
<tr>
<td>A7C0H</td>
<td>Integer48</td>
<td>rww</td>
</tr>
<tr>
<td>A800H</td>
<td>Unsigned56</td>
<td>rww</td>
</tr>
<tr>
<td>A840H</td>
<td>Integer56</td>
<td>rww</td>
</tr>
<tr>
<td>A880H</td>
<td>Integer64</td>
<td>rww</td>
</tr>
<tr>
<td>A8C0H</td>
<td>Unsigned64</td>
<td>rww</td>
</tr>
</tbody>
</table>

Table 5: Output network variables

This distribution described according to the /DSP-302/ description rules is given in the Appendix.

The entries MaxCnt, missing in the list above, depend on the available memory of the device. For a further simplification, the EDS files for devices following DS-405 are allowed to omit this description. This is marked by setting additionally the second bit of the entry DynamicChannelsSupported of section DeviceInfo:

```
[DeviceInfo]
....
DynamicChannelsSupported=3
```

### 5.1.3 Variable names

Object dictionary entries which have names that are no legal variable names for IEC 61131-3 shall not be usable to the IEC 61131-3 system. No automatic renaming is defined by this paper.
5.2 Function block based access

IEC 61131-3 Function blocks

This chapter describes standard function blocks to access CANopen from IEC 61131-3. All function blocks and data types defined in this chapter are optional in the sense of CANopen standardisation.

The following table can be used by IEC 61131-3 compliant systems to state the features covered:

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CIA405_DEVICE</td>
</tr>
<tr>
<td>2</td>
<td>CIA405_SDO_ERROR</td>
</tr>
<tr>
<td>3</td>
<td>CIA405_EMCY_ERROR</td>
</tr>
<tr>
<td>4</td>
<td>CIA405_STATE</td>
</tr>
<tr>
<td>5</td>
<td>CIA405_TRANSITION_STATE</td>
</tr>
<tr>
<td>6</td>
<td>CIA405_CANOPEN_KERNEL_ERROR</td>
</tr>
</tbody>
</table>

Table 6: New data types

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Function block</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CIA405_RECV_EMCY_DEV</td>
</tr>
<tr>
<td>2</td>
<td>CIA405_RECV_EMCY</td>
</tr>
<tr>
<td>3</td>
<td>CIA405_SDO_WRITE4</td>
</tr>
<tr>
<td>4</td>
<td>CIA405_SDO_WRITE7</td>
</tr>
<tr>
<td>5</td>
<td>CIA405_SDO_WRITE14</td>
</tr>
<tr>
<td>6</td>
<td>CIA405_SDO_WRITE21</td>
</tr>
<tr>
<td>7</td>
<td>CIA405_SDO_READ4</td>
</tr>
<tr>
<td>8</td>
<td>CIA405_SDO_READ7</td>
</tr>
<tr>
<td>9</td>
<td>CIA405_SDO_READ14</td>
</tr>
<tr>
<td>10</td>
<td>CIA405_SDO_READ21</td>
</tr>
<tr>
<td>11</td>
<td>CIA405_GET_LOCAL_NODE_ID</td>
</tr>
<tr>
<td>12</td>
<td>CIA405_GET_STATE</td>
</tr>
<tr>
<td>13</td>
<td>CIA405_GET_CANOPEN_KERNEL_STATE</td>
</tr>
<tr>
<td>14</td>
<td>CIA405_NMT</td>
</tr>
<tr>
<td>15</td>
<td>CIA405_SDO_WRITE</td>
</tr>
<tr>
<td>16</td>
<td>CIA405_SDO_READ</td>
</tr>
</tbody>
</table>

Table 7: Function blocks
5.2.1 General function block design issues

5.2.1.1 Naming conventions
A specific prefix shall be given to all function block and data type names defined herein. Currently ‘CIA405’ is used as that prefix.

5.2.1.2 Data types vs. data length
It is possible to represent transmission of data of arbitrary length within IEC 61131-3. However, the interfaces to functions blocks allowing for that may be quite more difficult to use and understand. Therefore different functions blocks are defined herein, for the transmission of a fixed (maximum) amount of data each as well as blocks for arbitrary length.

5.2.1.3 User defined data types
User defined data types are used to represent items of CANopen in IEC 61131-3.

5.2.1.4 Time out
The interface is designed such that it is possible to wrap the calls with a timer block to implement time-outs. Additionally, it is allowed that lower level CANopen software implement their own time-outs and report such as errors to the caller.

5.2.1.5 Additional parameters
Some systems may support or require additional information in the function blocks. For example a PLC may have several CAN channels. In that case the function blocks will require the channel number of the service. It is allowed to add these parameters to the function blocks.

Applications depending on that will always have to be adapted, if the environment is changed. So the freedom of additional parameters will not lead to really new compatibility/portability problems.

5.2.2 Data types
The following data types shall be used with the standard function blocks:

Type CIA405_DEVICE shall represent the Node ID of a device:

```
TYPE
  CIA405_DEVICE: USINT (0..127);
END_TYPE
```

Figure 4: Type CIA405_DEVICE

Type CIA405_SDO_ERROR shall represent error information as defined in /DS-301/:

```
TYPE
  CIA405_SDO_ERROR: UDINT;
END_TYPE
```

Figure 5: Type CIA405_SDO_ERROR

Type CIA405_EMCY_ERROR contains emergency error information, as specified in /DS-301/:

```
TYPE
  CIA405_EMCY_ERROR : STRUCT
    EMCY_ERROR_CODE : WORD;
    ERROR_REGISTER : BYTE;
    ERROR_FIELD : ARRAY [1..5] of BYTE;
  END_STRUCT;
END_TYPE
```

Figure 6: Type CIA405_EMCY_ERROR
Type CIA405_STATE describes the state of the CANopen network layer, as defined in /DS-301/. The states INIT, RESET_COMM, RESET_APP, PRE_OPERATIONAL, STOPPED, OPERATIONAL correspond to the same states in /DS-301/. The State UNKNOWN shall be used, if the actual state of the device is not known (in example, if no guarding of the device is performed). The state NOT_AVAIL shall be used, if it is known, that the device is not available (in example, if guarding is performed and the device does not answer).

```
TYPE
  CIA405_STATE : (INIT,
                  RESET_COMM,
                  RESET_APP,
                  PRE_OPERATIONAL,
                  STOPPED,
                  OPERATIONAL,
                  UNKNOWN,
                  NOT_AVAIL);
END_TYPE
```

Figure 7: Type CIA405_STATE

Type CIA405_TRANSITION_STATE describes the state transitions of the CANopen network layer, as defined in /DS-301/:

```
TYPE
  CIA405_TRANSITION_STATE : (START_REMOTE_NODE,
                             STOP_REMOTE_NODE,
                             ENTER_PRE_OPERATIONAL,
                             RESET_NODE,
                             RESET_COMMUNICATION);
END_TYPE
```

Figure 8: Type CIA405_TRANSITION_STATE

Type CIA405_CANOPEN_KERNEL_ERROR contains error information about the CANopen Kernel.

```
TYPE
  CIA405_CANOPEN_KERNEL_ERROR : WORD;
END_TYPE
```

Figure 9: Type CIA405_CANOPEN_KERNEL_ERROR

Description of the CIA405_CANOPEN_KERNEL_ERROR value range:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000h</td>
<td>no error</td>
</tr>
<tr>
<td>0001h</td>
<td>Other error (see further error registers)</td>
</tr>
<tr>
<td>0002h</td>
<td>Data overflow</td>
</tr>
<tr>
<td>0003h</td>
<td>Time out</td>
</tr>
<tr>
<td>0004h - 000Fh</td>
<td>Reserved for further SDO errors</td>
</tr>
<tr>
<td>0010h</td>
<td>CAN Bus off</td>
</tr>
<tr>
<td>0011h</td>
<td>CAN Error Passive</td>
</tr>
<tr>
<td>0012h - 001Fh</td>
<td>Reserved for further internal Kernel errors</td>
</tr>
</tbody>
</table>
5.2.3 SDO access

5.2.3.1 SDO write

```plaintext
FUNCTION_BLOCK CIA405_SDO_WRITE4

VAR_INPUT
  DEVICE : CIA405_DEVICE;
  INDEX : WORD;
  SUBINDEX : BYTE;
  ENABLE : BOOL;
  DATA : ARRAY [1..4] of BYTE;
  DATALENGTH : USINT;
END_VAR

VAR_OUTPUT
  CONFIRM  : BOOL := FALSE;
  ERROR : CIA405_CANOPEN_KERNEL_ERROR;
  ERRORINFO : CIA405_SDO_ERROR;
END_VAR

END_FUNCTION_BLOCK
```

Figure 10: Function block CIA405_SDO_WRITE4

Enable

1

3

Confirm

2

4

Figure 11: CIA405_SDO_WRITE4 typical timing diagram

Figure 11 shows a typical timing diagram for CIA405_SDO_WRITE4. After all data is provided to the inputs, ENABLE is set to TRUE 1. The SDO will be sent, and when the CANopen software reports success to the function block, output CONFIRM will be changed to TRUE 2. The caller will see this and change ENABLE to FALSE 3, which in turn will cause output CONFIRM to change to FALSE 4.

The specification is as follows:

1. With a rising edge on input ENABLE, the function block will sample the inputs and initiate the transmission of the SDO specified in DATA and DATALENGTH to the recipient specified with DEVICE, INDEX and SUBINDEX. The value of DEVICE is limited to a range of 1 to 127. For access to the local object dictionary it is allowed to use the value 0 for DEVICE.

2. With TRUE on input ENABLE, the function block is allowed to continue execution. If a result is reported on the call by lower level CANopen software, outputs CONFIRM, ERROR and ERRORINFO are set accordingly.

3. With a falling edge on input ENABLE the function block will terminate. If the transmission did not finish yet, it will be aborted if possible. Outputs CONFIRM and ERROR will both be set to FALSE respectively "no error".

4. With a FALSE on input ENABLE, the function block will return immediately and not take any action.

The result of a writing operation is reported (immediately after the writing call or afterwards) in outputs CONFIRM, ERROR and ERRORINFO with a rising edge on either CONFIRM or ERROR. In case of ERROR is equal to 1 (other error), ERRORINFO will give more specified information on the
cause. This is especially true for the occurrence of an SDO Abort. Then ERRORINFO contains the Abort code.

It cannot be assumed in general that this function can synchronously complete; rather, it should be possible to have this function continue to be executed while the call to CIA405_SDO_WRITE4 returns and the PLC program is continued. Therefore, the result of a call may be available only several cycles after a rising edge has been applied to ENABLE.

5.2.3.2 SDO write for different data lengths
In addition to CIA405_SDO_WRITE4, function blocks

- CIA405_SDO_WRITE7,
- CIA405_SDO_WRITE14,
- CIA405_SDO_WRITE21,
- etc.

may be implemented with the same interface as CIA405_SDO_WRITE4, except that the upper limit of the array on input DATA shall be changed to 7, 14, or 21 respectively. The first element of the array represents the first data byte of the data stream transmitted with an CIA405_SDO_WRITEx function block.

5.2.3.3 SDO write for arbitrary length
This function block can be used as alternative to the CIA_SDO_WRITEx functions blocks. It supports the arbitrary length of data.

```plaintext
FUNCTION_BLOCK CIA405_SDO_WRITE
  VAR_INPUT
    DEVICE : CIA405_DEVICE;
    INDEX : WORD;
    SUBINDEX : BYTE;
    ENABLE : BOOL;
    DATA : ANY
    DATALENGTH : USINT;
  END_VAR
  VAR_OUTPUT
    CONFIRM : BOOL := FALSE;
    ERROR : CIA405_CANOPEN_KERNEL_ERROR;
    ERRORINFO : CIA405_SDO_ERROR;
  END_VAR
END_FUNCTION_BLOCK
```

Figure 12: Function block CIA405_SDO_WRITE

The behaviour and timing is the same as with CIA405_SDO_WRITE4.
5.2.3.4 SDO read

```
FUNCTION_BLOCK CIA405_SDO_READ4

VAR_INPUT
  DEVICE : CIA405_DEVICE;
  INDEX : WORD;
  SUBINDEX : BYTE;
  ENABLE : BOOL;
END_VAR

VAR_OUTPUT
  DATA : ARRAY [1..4] of BYTE;
  DATALENGTH : USINT;
  CONFIRM : BOOL;
  ERROR : CIA405_CANOPEN_KERNEL_ERROR;
  ERRORINFO : CIA405_SDO_ERROR;
END_VAR

END_FUNCTION_BLOCK
```

Figure 13: Function block CIA405_SDO_READ4

Enable
[1] [2] [3] [4]

Confirm
[2] [3] [4]

Figure 14: CIA405_SDO_READ4 typical timing diagram

Figure 14 shows a typical timing diagram for CIA405_SDO_READ4. After all data is provided to the inputs, ENABLE is set to TRUE [1]. The SDO will be received, and when the CANopen software reports success to the function block, output CONFIRM will be changed to TRUE [2]. The caller will see this and change ENABLE to FALSE [3], which in turn will cause output CONFIRM to change to FALSE [4].

The specification is as follows:

1. With a rising edge on input ENABLE, the function block will sample the inputs and initiate the transmission of the SDO specified with DEVICE, INDEX and SUBINDEX. The value of DEVICE is limited to a range of 1 to 127. For access to the local object dictionary it is allowed to use the value 0 for DEVICE.

2. With TRUE on input ENABLE, the function block is allowed to continue execution. If a result is reported on the call by lower level CANopen software, outputs CONFIRM, DATA and DATALENGTH (in case of success) or ERROR and ERRORINFO (in case of failure) are set accordingly.

3. With a falling edge on input ENABLE the function block will terminate. If the transmission did not finish yet, it will be aborted if possible. Outputs CONFIRM and ERROR will both be set to FALSE respectively "no error".

4. With a FALSE on input ENABLE, the function block will return immediately and not take any action.

Like with CIA405_SDO_WRITE4, it cannot be assumed that this function can be completed before the call to CIA405_SDO_READ4 returns. Therefore, the result may be available several cycles after the receiving call.
5.2.3.5 SDO read for different data lengths
In addition to CIA405_SDO_READ4, function blocks

- CIA405_SDO_READ7,
- CIA405_SDO_READ14,
- CIA405_SDO_READ21,
- etc.

may be implemented with the same interface as CIA405_SDO_READ4, except that the upper limit of the array on output DATA shall be changed to 7, 14, or 21 respectively. The first element of the array represents the first data byte of the data stream transmitted with an CIA405_SDO_READx function block.

If a function block receives more data than specified, the ERROR output signals this with the error code 0002h ("data overflow"). It is application dependent to use this fragment of data or not.

5.2.3.6 SDO read for arbitrary length
This function block can be used as alternative to the CIA_SDO_READx functions blocks. It supports the arbitrary length of data.

FUNCTION_BLOCK CIA405_SDO_READ
VAR_INPUT
  DEVICE : CIA405_DEVICE;
  INDEX : WORD;
  SUBINDEX : BYTE;
  ENABLE : BOOL;
END_VAR
VAR_OUTPUT
  DATA : ANY;
  DATALENGTH : USINT;
  CONFIRM : BOOL;
  ERROR : CIA405_CANOPEN_KERNEL_ERROR;
  ERRORINFO : CIA405_SDO_ERROR;
END_VAR
END_FUNCTION_BLOCK

Figure 15: Function block CIA405_SDO_READ

The behaviour and timing is the same as with CIA405_SDO_READ4.

5.2.4 Other function blocks

5.2.4.1 Own node id
Function block CIA405_GET_LOCAL_NODE_ID returns the own node ID.

FUNCTION_BLOCK CIA405_GET_LOCAL_NODE_ID
VAR_INPUT
  ENABLE : BOOL;
END_VAR
VAR_OUTPUT
  CONFIRM : BOOL;
  DEVICE : CIA405_DEVICE;
END_VAR
END_FUNCTION_BLOCK

Figure 16: Function block CIA405_GET_LOCAL_NODE_ID
5.2.4.2 Query state

Function block CIA405_GET_STATE returns the current state of a CANopen network device:

```plaintext
FUNCTION_BLOCK CIA405_GET_STATE
    VAR_INPUT
        DEVICE : CIA405_DEVICE;
        ENABLE : BOOL;
    END_VAR
    VAR_OUTPUT
        CONFIRM : BOOL;
        STATE : CIA405_STATE;
    END_VAR
END_FUNCTION_BLOCK
```

Figure 17: Function block CIA405_GET_STATE

The usage and behaviour of ENABLE and CONFIRM is the same like with CIA405_SDO_WRITE4. Refer to Figure 11. If the state is not known, the function block returns with the state UNKNOWN. This will occur, if the device is not guarded and Heartbeat is not running. If Guarding is performed and the device does not answer or it is known for other reasons, that the device is not available, the functions returns the state NOT_AVAIL. The value of DEVICE is limited to a range of 1 to 127. If DEVICE is 0 or equal to the own Node-ID, the function block returns the state of the local communication process.

Function block CIA405_GET_CANOPEN_KERNEL_STATE returns the current state of the CANopen Kernel:

```plaintext
FUNCTION_BLOCK CIA405_GET_CANOPEN_KERNEL_STATE
    VAR_INPUT
        ENABLE : BOOL;
    END_VAR
    VAR_OUTPUT
        CONFIRM : BOOL;
        STATE : CIA405_CANOPEN_KERNEL_ERROR;
    END_VAR
END_FUNCTION_BLOCK
```

Figure 18: Function block CIA405_GET_CANOPEN_KERNEL_STATE

The usage and behaviour of ENABLE and CONFIRM is the same like with CIA405_GET_STATE.

5.2.4.3 Network management

Function block CIA405_NMT controls network management functions of one or all CANopen nodes:

```plaintext
FUNCTION_BLOCK CIA405_NMT
    VAR_INPUT
        DEVICE : CIA405_DEVICE;
        STATE : CIA405_TRANSITION_STATE;
        ENABLE : BOOL;
    END_VAR
    VAR_OUTPUT
        CONFIRM : BOOL;
        ERROR : CIA405_CANOPEN_KERNEL_ERROR;
    END_VAR
END_FUNCTION_BLOCK
```

Figure 19: Function block CIA405_NMT

Within this function block it is allowed to assign DEVICE the value 0. This means that all nodes of the network will enter the selected state.
5.2.4.4 Receive emergency object from a specific device

```plaintext
FUNCTION_BLOCK CIA405_RECV_EMCY_DEV
VAR_INPUT
  DEVICE : CIA405_DEVICE;
  ENABLE : BOOL;
END_VAR
VAR_OUTPUT
  CONFIRM : BOOL;
  ERROR : CIA405_CANOPEN_KERNEL_ERROR;
  ERRORINFO : CIA405_EMCY_ERROR;
END_VAR
END_FUNCTION_BLOCK
```

Figure 20: Function block CIA405_RECV_EMCY_DEV

CIA405_RECV_EMCY_DEV will check if an emergency object (/DS-301/) has been received from DEVICE. If so, and no error occurred, output CONFIRM is changed to TRUE and ERROR to 0. If an error occurred (either during reception of this emergency object or without such an emergency object arriving), CONFIRM is set to FALSE and ERROR is set to the responding error value and the cause is specified in ERRORINFO. One emergency object shall be reported only once. The value of DEVICE is limited to a range of 1 to 127.

5.2.4.5 Emergency object from any device

```plaintext
FUNCTION_BLOCK CIA405_RECV_EMCY
VAR_INPUT
  ENABLE : BOOL;
END_VAR
VAR_OUTPUT
  CONFIRM : BOOL;
  DEVICE : CIA405_DEVICE;
  ERROR : CIA405_CANOPEN_KERNEL_ERROR;
  ERRORINFO : CIA405_EMCY_ERROR;
END_VAR
END_FUNCTION_BLOCK
```

Figure 21: Function block CIA405_RECV_EMCY

CIA405_RECV_EMCY will check if an emergency object (/DS-301/) has been received from any device. If so, and no error occurred, output CONFIRM is changed to TRUE and ERROR to 0 and DEVICE is set to the ID of the device sending the emergency object. If an error occurred (either during reception of this emergency object or without such an emergency object arriving), CONFIRM is set to FALSE and ERROR is set to the responding error and the cause is specified in ERRORINFO. One emergency object shall be reported only once.
6 Tool integration

This chapter defines the information exchange between software packages concerning the network variables. Typical software packages in this context are the IEC 61131-3 programming system and network configuration / network project planning system. Both systems may be integrated in one software. In that case, the information exchange can be handled internally. In the other case there exist two separate software packages, that have the task to exchange some required information in a standardized way:

![Diagram of IEC61131 and CANopen Network Configuration](image)

Figure 22: Correlation between programming and network configuration

6.1 Basic concept

The described mechanism is based on the assumption that at one time only one tool is active. This means that only one tool has the right to write the information to the files. On a multitasking operating system for example all tools can run at the same time but only one tool is allowed to modify the project data by writing to the project files at the same time.

Further it is assumed that all necessary configuration information of a CANopen network is stored in DCF files (file based operating system). These files and formats are already defined within /DS-301/ specification. All used tools must be able to have read and write access to these EDS and DCF files. It is necessary that each tool works with the same files at one location. The way the access is handled depends on the used operating system.

In order to specify all nodes appearing in a network a special file called „Nodelist“ shall be created. The format of this file is the same as the format of an EDS/DCF file (Windows ini file). To manage the read/write access to the DCF files one more file called „Access Handler“ shall be created.

Whenever a tool wants to modify the files „Nodelist“, „Access Handler“ and DCF it shall check for existence first.

Refer chapter 8.4.4 for application notes.

6.2 Node list

This file contains all nodes within a network. The way this file is accessed by the tools depends on the used operating system. For a file based system this means to show the tools the path and the filename. This location must be the same where all DCF files are stored.

With the minimum information of the „Nodelist“ a tool is able to ask the user for each node’s DCF, EDS and how to establish a read or write access. Further entries are for support of more than one network.

To allow tools an automatic access to the „Nodelist“ the following filename is specified:

„nodelist.cpj“
The following entries are specified:

<table>
<thead>
<tr>
<th>[Sections] and entries</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Topology]</td>
<td>Optional, if this section is missing the tools can assume that there is at least one network with the number 1.</td>
</tr>
<tr>
<td>NetName =</td>
<td>Optional name for a net. This is helpful for a tool to give the user assistance by identifying the net.</td>
</tr>
<tr>
<td>Nodes = 0x02</td>
<td>Mandatory, number of nodes within the net. The range of the number is from 1 to 127. The number is to be read as a hexadecimal number with leading 0x.</td>
</tr>
<tr>
<td>NodeXPresent = 0x01</td>
<td>Mandatory for each existing node. X specifies the CANopen Node number within the network. X is coded decimal with no leading zeros (e.g. Node1Present, Node10Present, Node127Present). A value of 0x01 means the node X is present. A value of 0x00 or missing entry means the node is not present. All other values are reserved.</td>
</tr>
<tr>
<td>NodeXName = Node1</td>
<td>Optional node name. This is for tools to help the user to identify the node. X is coded decimal with no leading zeros (e.g. Node1Name, Node10Name, Node127Name).</td>
</tr>
<tr>
<td>NodeXDCFName = Node1.dcf</td>
<td>Optional DCF filename for this node. The filename should be written without path information. This entry is helpful for tools based on a file system. X is coded decimal with no leading zeros (e.g. Node1DCFName, Node10DCFName, Node127DCFName).</td>
</tr>
<tr>
<td>EDSBaseName = path</td>
<td>Optional path to the EDS files. This entry is helpful for tools based on a file system.</td>
</tr>
</tbody>
</table>

Table 9: Entries in Nodelist file

Further sections and entries may exist. It is the responsibility of the tool manufacturers to avoid collisions in the naming of sections and entries. For the syntax definition of section, entry and the entry values refer /DSP-306/.

### 6.3 Access handler

The file „Access Handler“ handles the read and write access to the DCF files. The „Access Handler“ is a simple file without sections and entries. The only information is the name of the locking tool. If a tool wants to get the write access it can open the file exclusively. After writing to the DCF files the tool has to release the file. If a tool was blocked by another it can inform the user with detailed information about the locking tool.

The filename of this „Access Handler“ is „lock“.

<table>
<thead>
<tr>
<th>Content</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>„Name of the tool“</td>
<td>Optional, name of the tool that has the read/write access.</td>
</tr>
</tbody>
</table>
7 Utility functions

7.1 Remote functions between resources

7.1.1 Down-/Upload
The program download mechanism is defined in /DSP-302/.

7.2 Object dictionary entries
Modern programming systems provide users with project management, device configuration, test and debug units. For testing and debugging the following entries are defined, which can be evaluated by tools (e.g. configuration, SCADA systems). The following approach is a combination of the CANopen project and the IEC 61131 configuration hierarchy. A CANopen project consists of at least one or more PLCs in one network.

![Figure 23: Project structure](image)

Objects defined in this chapter, that have the data type visible string use strings with the fixed length of 32 characters. The character representation is used from the IEC 61131-3 chapter 2.1.

For version numbers the data type unsigned32 is used. The higher word includes the major number and the lower word the minor number, both are BCD coded.

The following object entries have to support the read access, the write access is optional.

To avoid consistence conflicts with CANopen tools during network configuration (download) the following objects should support the „ObjFlags“ entry in the EDS/DCF with the „Refuse write on download“ bit (bit 0) set. Imagine a CANopen tool writes a configuration over the CAN bus to a node. For the tool it is not possible to suppress explicit writing on the following entries. But this objects are handled generally by the IEC 61131 programming tool. So it is not guaranteed that the information in the DCF matches these configuration. Note the solution with the „ObjFlags“ entry specified in /DSP-306/.

7.2.1 Project name
Typical network projects consist of at least one configuration. The assignment between configurations and projects could be done by the project name.
Object Description

<table>
<thead>
<tr>
<th>Index</th>
<th>9800H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Project_Name</td>
</tr>
<tr>
<td>Object Code</td>
<td>VAR</td>
</tr>
<tr>
<td>Data Type</td>
<td>Visible String</td>
</tr>
</tbody>
</table>

Value Description

| Object Class | Optional |
| Access | R (optional W) |
| PDO Mapping | No |
| Value Range | Visible String |
| Mandatory Range | No |
| Default Value | No |

7.2.2 Configuration

The configuration includes a list of all resources used in a PLC system. A configuration will be identified by its name and version.

Configuration definition structure

<table>
<thead>
<tr>
<th>Index</th>
<th>Subindex</th>
<th>Field in Configuration_Def Record</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0080H</td>
<td>0H</td>
<td>Number of supported entries</td>
<td>Unsigned8</td>
</tr>
<tr>
<td></td>
<td>1H</td>
<td>Configuration_Name</td>
<td>Visible String</td>
</tr>
<tr>
<td></td>
<td>2H</td>
<td>Configuration_Version</td>
<td>Unsigned32</td>
</tr>
<tr>
<td></td>
<td>3H</td>
<td>Number_Of_Resources</td>
<td>Unsigned8</td>
</tr>
</tbody>
</table>

Detailed object description

<table>
<thead>
<tr>
<th>Index</th>
<th>9501H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Configuration</td>
</tr>
<tr>
<td>Object Code</td>
<td>RECORD</td>
</tr>
<tr>
<td>Number of Elements</td>
<td>3</td>
</tr>
<tr>
<td>Data Type</td>
<td>Configuration_Def</td>
</tr>
</tbody>
</table>

Value description

| Sub-Index | 0H |
| Description | Number of entries |
| Object class | Optional |
| Access | RO |
| PDO Mapping | No |
| Value Range | 1H - 3H |
| Mandatory Range | No |
| Default Value | 3H |

| Sub-Index | 1H |
| Description | Configuration_Name |
| Object class | Optional |
| Access | R (optional W) |
| PDO Mapping | no |
| Value Range | Visible String |
| Mandatory Range | No |
7.2.3 Resources

The declaration of resources provides a mechanism to allocate tasks and programs to a resource (e.g., PLC CPU). A resource will be identified by its name and version. The task handle is a reference to the first task on a resource.

Resource definition structure

<table>
<thead>
<tr>
<th>Index</th>
<th>Subindex</th>
<th>Field in Resource_Def Record</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0081H</td>
<td>0H</td>
<td>Number of supported entries</td>
<td>Unsigned8</td>
</tr>
<tr>
<td></td>
<td>1H</td>
<td>Resource_Name</td>
<td>Visible String</td>
</tr>
<tr>
<td></td>
<td>2H</td>
<td>Resource_Version</td>
<td>Unsigned32</td>
</tr>
<tr>
<td></td>
<td>3H</td>
<td>Number_Of_Tasks</td>
<td>Unsigned8</td>
</tr>
<tr>
<td></td>
<td>4H</td>
<td>Task_Handle</td>
<td>Unsigned32</td>
</tr>
</tbody>
</table>

Detailed object description

Index: 9600H - 96FFH
Name: Resource
Object Code: RECORD
Number of Elements: 4
Data Type: Resource_Def

Value description

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>Number of supported entries</td>
<td>Optional</td>
<td>RO</td>
<td>No</td>
<td>Unsigned8</td>
<td>4</td>
<td>No</td>
</tr>
</tbody>
</table>

Sub-Index: 1H
7.2.4 Task
A task will be identified by its name or identifier. The task types cyclic, event and timertask are supported.

Representation of the Values: cyclic = 0x0, event = 0x1, timertask = 0x2

Task definition structure

<table>
<thead>
<tr>
<th>Index</th>
<th>Subindex</th>
<th>Field in Task_Def Record</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0082H</td>
<td>0H</td>
<td>Number of supported entries</td>
<td>Unsigned8</td>
</tr>
<tr>
<td></td>
<td>1H</td>
<td>Task_Name</td>
<td>Visible String</td>
</tr>
<tr>
<td></td>
<td>2H</td>
<td>Task_Identifier</td>
<td>Unsigned32</td>
</tr>
<tr>
<td></td>
<td>3H</td>
<td>Task_Types</td>
<td>Unsigned32</td>
</tr>
<tr>
<td></td>
<td>4H</td>
<td>Task_Priority</td>
<td>Unsigned32</td>
</tr>
<tr>
<td></td>
<td>5H</td>
<td>Time_Intervall</td>
<td>Unsigned32</td>
</tr>
</tbody>
</table>

Detailed object description

<table>
<thead>
<tr>
<th>Index</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>9700H - 97FFH</td>
<td>Task</td>
</tr>
</tbody>
</table>
### Object Code

<table>
<thead>
<tr>
<th>Number of Elements</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Task_Def</td>
</tr>
</tbody>
</table>

#### Value description

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0H</td>
<td>Number of supported entries</td>
<td>Optional</td>
<td>RO</td>
<td>No</td>
<td>Unsigned8</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1H</td>
<td>Task_Name</td>
<td>Optional</td>
<td>R (optional W)</td>
<td>No</td>
<td>Visible String</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2H</td>
<td>Task_Identifier</td>
<td>Optional</td>
<td>R (optional W)</td>
<td>No</td>
<td>Unsigned32</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3H</td>
<td>Task_Type</td>
<td>Optional</td>
<td>R (optional W)</td>
<td>No</td>
<td>Unsigned32</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
<th>Object class</th>
<th>Access</th>
<th>PDO Mapping</th>
<th>Value Range</th>
<th>Mandatory Range</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>4H</td>
<td>Task_Priority</td>
<td>Optional</td>
<td>R (optional W)</td>
<td>No</td>
<td>Unsigned32</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sub-Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5H</td>
<td>Time_Interval</td>
</tr>
<tr>
<td>Object class</td>
<td>Optional</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>Access</td>
<td>R (optional W)</td>
</tr>
<tr>
<td>PDO Mapping</td>
<td>No</td>
</tr>
<tr>
<td>Value Range</td>
<td>Unsigned32</td>
</tr>
<tr>
<td>Mandatory Range</td>
<td>No</td>
</tr>
<tr>
<td>Default Value</td>
<td>No</td>
</tr>
</tbody>
</table>

7.2.5 Start/Stop (Program/Task)
see /DSP-302/

7.2.6 Debugging/Monitoring
see /DSP-302/
8 Appendix (informative)

8.1 Detailed description of network variable segments

[DynamicChannels]
NrOfSeg=36
; -----
Type1=1
Dir1=ro
Range1=0xA080-0xA0BF
PPOffset1=0, 1
; -----
Type2=2
Dir2=ro
Range2=0xA000-0xA03F
PPOffset2=0
; -----
Type3=5
Dir3=ro
Range3=0xA040-0xA07F
PPOffset3=0
; -----
Type4=3
Dir4=ro
Range4=0xA0C0-0xA0FF
PPOffset4=0
; -----
Type5=6
Dir5=ro
Range5=0xA100-0xA13F
PPOffset5=0
; -----
Type6=10
Dir6=ro
Range6=0xA140-0xA17F
PPOffset6=0
; -----
Type7=16
Dir7=ro
Range7=0xA180-0xA1BF
PPOffset7=0
; -----
Type8=4
Dir8=ro
Range8=0xA1C0-0xA1FF
PPOffset8=0
; -----
Type9=2
Dir9=ro
Range9=0xA200-0xA23F
PPOffset9=0
; -----
Type10=12
Dir10=ro
Range10=0xA2C0-0xA2FF
PPOffset10=0
; -----
Type11=18
Dir11=ro
Range11=0xA280-0xA2BF
PPOffset11=0
; -----
Type12=13
Dir12=ro
Range12=0xA340-0xA37F
PPOffset12=0
; -----
Type13=19
Dir13=ro
Range13=0xA300-0xA33F
PPOffset13=0
; -----
Type14=14
Dir14=ro
Range14=0xA3C0-0xA3FF
PPOffset14=0
; -----
Type15=1A
Dir15=ro
Range15=0xA380-0xA3BF
PPOffset15=0
; -----
Type16=1B
Dir16=ro
Range16=0xA400-0xA43F
PPOffset16=0
; -----
Type17=1C
Dir17=ro
Range17=0xA440-0xA47F
PPOffset17=0
; -----
Type18=1D
Dir18=ro
Range18=0xA240-0xA27F
PPOffset18=0
; -----
Type19=1E
Dir19=ro
Range19=0xA500-0xA53F
PPOffset19=0, 1
; -----
Type20=2
Dir20=ro
Range20=0xA480-0xA4BF
PPOffset20=0
; -----
Type21=5
Dir21=ro
Range21=0xA4C0-0xA4FF
PPOffset21=0
; -----
Type22=3
Dir22=ro
Range22=0xA540-0xA57F
PPOffset22=0
; -----
Type23=6
Dir23=ro
Range23=0xA580-0xA5BF
PPOffset23=0
; -----
Type24=10
Dir24=ro
Range24=0xA5C0-0xA5FF
PPOffset24=0

; -----
Type25=16
Dir25=ro
Range25=0xA600-0xA63F
PPOffset25=0
; -----
Type26=4
Dir26=ro
Range26=0xA640-0xA67F
PPOffset26=0
; -----
Type27=7
Dir27=ro
Range27=0xA680-0xA6BF
PPOffset27=0
; -----
Type28=12
Dir28=ro
Range28=0xA740-0xA77F
PPOffset28=0
; -----
Type29=18
Dir29=ro
Range29=0xA700-0xA73F
PPOffset29=0
; -----
Type30=13
Dir30=ro
Range30=0xA7C0-0xA7FF
PPOffset30=0
; -----
Type31=19
Dir31=ro
Range31=0xA780-0xA7BF
PPOffset31=0
; -----
Type32=14
Dir32=ro
Range32=0xA840-0xA87F
PPOffset32=0
; -----
Type33=1A
Dir33=ro
Range33=0xA900-0xA93F
PPOffset33=0
; -----
Type34=15
Dir34=ro
Range34=0xA880-0xA8BF
PPOffset34=0
; -----
Type35=1B
Dir35=ro
Range35=0xA8C0-0xA8FF
PPOffset35=0
; -----
Type36=8
Dir36=ro
Range36=0xA6C0-0xA6FF
PPOffset36=0

© CiA 2008 - All rights reserved
8.2 IEC 61131-3 object dictionary overview

The following table shows all objects, affected by these specification.

<table>
<thead>
<tr>
<th>Index (hex)</th>
<th>Object</th>
<th>Name</th>
<th>Type</th>
<th>Acc.</th>
<th>M/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>0080H</td>
<td>DEFSTRUCT</td>
<td>Configuration_Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0081H</td>
<td>DEFSTRUCT</td>
<td>Resource_Def</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0082H</td>
<td>DEFSTRUCT</td>
<td>Task_Def</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000H</td>
<td>VAR</td>
<td>Device type</td>
<td>Unsigned32</td>
<td>RO</td>
<td>M</td>
</tr>
<tr>
<td>9501H</td>
<td>RECORD</td>
<td>Configuration</td>
<td>Configuration_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>9600H</td>
<td>RECORD</td>
<td>1st Resource</td>
<td>Resource_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>9601H</td>
<td>RECORD</td>
<td>2nd Resource</td>
<td>Resource_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96FFH</td>
<td>RECORD</td>
<td>255th Resource</td>
<td>Resource_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>9700H</td>
<td>RECORD</td>
<td>1st Task</td>
<td>Task_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>9701H</td>
<td>RECORD</td>
<td>2nd Task</td>
<td>Task_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97FFH</td>
<td>RECORD</td>
<td>255th Task</td>
<td>Task_Def</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>9800H</td>
<td>VAR</td>
<td>Project_Name</td>
<td>Visible String</td>
<td>RW</td>
<td>O</td>
</tr>
<tr>
<td>A000H</td>
<td>ARRAY</td>
<td>1st Dynamic Integer8</td>
<td>Integer8</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A03FH</td>
<td>ARRAY</td>
<td>64th Dynamic Integer8</td>
<td>Integer8</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td>A040H</td>
<td>ARRAY</td>
<td>1st dynamic Unsigned8</td>
<td>Unsigned8</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A07FH</td>
<td>ARRAY</td>
<td>64th Dynamic Unsigned8</td>
<td>Unsigned8</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td>A080H</td>
<td>ARRAY</td>
<td>1st Dynamic Boolean</td>
<td>Boolean</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0BFH</td>
<td>ARRAY</td>
<td>64th Dynamic Boolean</td>
<td>Boolean</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td>A0C0H</td>
<td>ARRAY</td>
<td>1st Dynamic Integer16</td>
<td>Integer16</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0FF0H</td>
<td>ARRAY</td>
<td>64th Dynamic Integer16</td>
<td>Integer16</td>
<td>RO</td>
<td>O</td>
</tr>
<tr>
<td>A100H</td>
<td>ARRAY</td>
<td>1st Dynamic Unsigned16</td>
<td>Unsigned16</td>
<td>RO</td>
<td>O</td>
</tr>
</tbody>
</table>
### Table 10: List of object dictionary entries

<table>
<thead>
<tr>
<th></th>
<th>Array</th>
<th>64&lt;sup&gt;th&lt;/sup&gt; Dynamic</th>
<th>Element Type</th>
<th>Access</th>
<th>Trend</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A13FH</td>
<td>ARRAY</td>
<td>Dynamic Unsigned16</td>
<td>Unsigned16</td>
<td>RO</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A440H</td>
<td>ARRAY</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; Dynamic</td>
<td>Unsigned64</td>
<td>RO</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A47FH</td>
<td>ARRAY</td>
<td>64&lt;sup&gt;th&lt;/sup&gt; Dynamic</td>
<td>Unsigned64</td>
<td>RO</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A480H</td>
<td>ARRAY</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; Dynamic</td>
<td>Integer8</td>
<td>RWW</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A4BFH</td>
<td>ARRAY</td>
<td>64&lt;sup&gt;th&lt;/sup&gt; Dynamic</td>
<td>Integer8</td>
<td>RWW</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A8C0H</td>
<td>ARRAY</td>
<td>1&lt;sup&gt;st&lt;/sup&gt; Dynamic</td>
<td>Unsigned64</td>
<td>RWW</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>A8FFH</td>
<td>ARRAY</td>
<td>64&lt;sup&gt;th&lt;/sup&gt; Dynamic</td>
<td>Unsigned64</td>
<td>RWW</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
8.3 Example DCF file

This is not a complete DCF. Only several specific entries for the DS405 are listed.

```
[DeviceInfo]
...
DynamicChannelsSupported=3
...

[Dyn}amicChannels]; optional since DynamicChannelsSupported=3
NrOfSeg=36 ; see application notes
Type1=1
Dir1=ro
Range1=0xA080-0xA0BF
PPOffset1=0,1
...
Type36=8
Dir36=rww
Range36=0xA6C0-0xA6FF
PPOffset36=0
...

[OptionalObjects]
1=0xA000
...

[A000]; Integer8 RO
SubNumber=3
ParameterName=Integer8_RO_Variables
ObjectType=8

[A000Sub0]
ParameterName=NROfSupportedObjects
ObjectType=0x7
DataType=0x0005
AccessType=ro
DefaultValue=2
PDOMapping=0

[A000Sub1]
ParameterName=<NameOfProcessVariable1>
ObjectType=0x7
DataType=0x0002
AccessType=ro
DefaultValue=
PDOMapping=1

[A000Sub2]
ParameterName=<NameOfProcessVariable2>
ObjectType=0x7
DataType=0x0002
AccessType=ro
DefaultValue=
PDOMapping=1
...
```

8.4 Application notes

8.4.1 Network variables

The usage of the dynamic index assignment of the network variables implies, that the object dictionary area contains arrays with gaps. For example an array may have two sub-objects 1 and 5.
8.4.2 Data type representation issues

In chapter 4.2 the internal data type representation is specified to be partially CANopen non-
compliant. If this happens, there may be different data representation of the same data values.

Example:

A PLC may have a byte ordering opposite to CANopen. When transferring a network variable of
data type Unsigned16 with value 1 via a PDO according to 4.2 the process picture will contain the
following value:

<table>
<thead>
<tr>
<th>Application value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001_1 = _1d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process image</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDO</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>abc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interpretation by other devices according to CANopen encoding rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001_1 = _1d</td>
</tr>
</tbody>
</table>

Figure 24: Data representation with PDO on CANopen non-compliant PLCs

When transferring the same value via SDO, there is another situation:

<table>
<thead>
<tr>
<th>Application value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001_1 = _1d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process image</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 0</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protocol stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDO</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>xyz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interpretation by other devices according to CANopen encoding rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100_2 = _256d</td>
</tr>
</tbody>
</table>

Figure 25: Data representation with SDO on CANopen non-compliant and compliant PLCs

It is the responsibility of the application / user to consider this. The manufacturer of a PLC may
decide to make the byte-re-ordering as an own task upon the protocol stack. This will avoid the
problem, but will need much more system resources.
8.4.3 EDS

The usage of "DynamicChannelsSupported=3" implies not to support all possible CANopen data types with a device. If the device supports only some of these data types (e.g. BOOL, 1-, 2- and 4 Byte wide types), this can be coded by using the section "DynamicChannels".

It is not allowed to redefine the section "DynamicChannels" of a 405 Profile. Tools are allowed to presuppose the objects as defined in these specification and it is recommended that tools, support the 405 Profile, do not interpret data types other than specified. A reason to enter the section "DynamicChannels" in a EDS/DCF is to support tools without knowledge of the 405 Profile (e.g. DSP-302 Tools) or to give information, what data types are supported or to supply the information of the implemented size of the segments.

For these reasons it is strongly recommended to include the section “DynamicChannels” in the EDS.

8.4.4 Tool integration

This mechanism allows all kind of tools to use the Project files. Users can work with programming environments, debugging tools and project planning tools or only simple node specific configuration tools. Each tool has full access to all information. Further there is no consistency problem between different files.

The locking mechanism is available on every disk operating system used in practice.

Definition of variables may be done in the Programming System. In that case the Network Configuration has to be informed about the name, type and address of these variables, since the Network Configuration is responsible for setting-up the appropriate communication channels for transferring the data contents. The other possibility is the generation of variables in the Network Planning System. This is useful on setting-up a network with distributed intelligence, where the interface between the separate processes has to be defined. In that case a possibility is required to transfer information about the generated variables to the Programming System of each programmable device of that network.

Chapter 6 specifies the data exchange via DCF according to Figure 26.

![Figure 26: Information exchange via DCF](image)

Some programming systems may not support DCF file formats. In practice it even may be not possible to extend them. But normally they will provide a possibility of data exchange in any other file format. For example this may be something like an include file written in IEC 61131-3 syntax itself (such as DTY). For fulfilling the specification in chapter 6 it is possible to use a converter module as shown in Figure 27.

![Figure 27: Data exchange using converter module](image)
Version 1.0 of DSP-405 defined mechanisms for data exchange via DCF and a file format called NVX. For compatibility reasons the usage of NVX can be interpreted in a way, that the converter is part of the programming system as well as of the Network Configuration System. This is illustrated in Figure 28.

![Figure 28: Compatibility for NVX format](image)

To allow the denotation of a device specific tool already by the device manufacturer the EDS may contain a description, which tools are to be used. On creating a DCF from that EDS, this tool description will be copied.

The description start with a section ```[Tools]``` containing the entry ```Items``` with the number of tools supported. Each tool is described in a section ```[Toolx]``` with x as decimal counter (1..Items). These sections contain the entries ```Name``` and ```Command``` giving the symbolic name and the concrete command. Further entries may exist. Manufacturers are responsible to avoid naming collisions. Command line parameters are not specified here. They depend on the converters.

The example shows a converter

```
[Tools]
Items=1

[Tool1]
Name=Convert DCF to DTY
Command=DCF2DTY $DCF $NAME.DTY
```
8.5 IEC 61131-3 sample code

To give a better impression of how the IEC 61131-3 defined items will be used in a typical application, here is one (non-normative) sample program:

```plaintext
PROGRAM HeatControl
VAR_EXTERN
    (* How to map variables between process image and CANopen will *)
    (* be very system dependent, this is just one possible way     *)
    Temperature: CIA405INT24;
    VentilatorSpeed: CIA405UINT48;
END_VAR
VAR_GLOBAL
    MaxTemperature: DINT := 21;
    Slow : ULINT := 300;    (* whatever unit *)
    Fast : ULINT := 800;    (* same unit     *)
END_VAR

LD Temperature
    CIA405INT24_TO_DINT
    GT MaxTemperature
    JMPC cooler

    (* seems not to hot, so run ventilator slow *)
    LD Slow
    ULINT_TO_CIA405UINT48
    ST VentilatorSpeed
    JMP go_on

cooler:
    (* seems to be hot, run ventilator faster *)
    LD Fast
    ULINT_TO_CIA405UINT48
    ST VentilatorSpeed

go_on:
    (* rest of program *)
END_PROGRAM
```

8.6 Implementation models for IEC 61131-3 data type support

Data types CIA405INT24 etc. as mentioned before are not standard with IEC 61131-3. How these are made available to the user is left to implementation, the following chapter describes several possible options. All these models are non-normative, serving only for a better understanding.

8.6.1 Native support

One possible implementation is to integrate support for these data types defined above into the IEC 61131-3 programming system. No conversions should be necessary then, the new data types can be integrated into the IEC 61131-3 hierarchy of data types, allowing all built-in instructions and many overloaded system functions to be used for these data types (ADD, EQ, MAX, ...)

Problem: Some effort to implement.

8.6.2 Padding to next best match

The new data types may be defined by using the closest matching IEC 61131-3 native type, e.g.

```plaintext
TYPE CIA405INT48: LINT; END_TYPE;
```

Problem: Memory space is wasted. Value of IEC 61131-3 may exceed limits of CANopen representation. Which instance shall handle this and how? Necessary padding bytes might impose problems on IEC 61131-3 memory mapping (e.g. process image).
8.6.3 Using arrays or struct

The new types may be defined using a array (or structure) of individual bytes:

```
TYPE CIA405INT48: array[1..6] of BYTE; END_TYPE;
```

or

```
TYPE CIA405INT48: STRUCT b1,b2,b3,b4,b5,b6: BYTE; END_TYPE;
```

Applications using only data types and conversion functions as defined in chapters before can be truly portable to different systems, no matter which of these three implementation models be used. Applications using details depending on any of these implementations (applying an ADD to one of these types, accessing individual elements of this array or relying on out-of-range behaviour) might not be portable.