CANopen device configuration editors

Dr. Heikki Saha

Author

Dr. Heikki Saha
TK Engineering
P.O. Box 810
FI-65101 Vaasa
Tel.: +358-6-35763-00
Fax: +358-6-35763-20
info@tke.fi

Links
www.tke.fi

Introduction

This article presents the main principles of device configuration editors as two case examples. The editor in the first example is targeted at CANopen encoders. It was possible to implement a generic editor because the encoders share a uniform logical structure [14]. The second example presents an editor for general purpose I/O devices. Because of the generality, device or device family specific editors are required [13]. Both device profiles are supported by a large number of devices, which helped to find out the main commonalities and differences.

In modern, distributed control systems re-use of functions expects the use of a limited number of components in as many instances as possible. Each component instance requires unique settings assigned during the system design. A common misunderstanding is that CANopen is only a serial communication protocol and therefore parameterization is considered a part of software development. Such an approach easily leads into system configuration inconsistencies, because system level configuration management is missing. Standard DCF files are used for storing the settings of CANopen devices. The simplest approach is to manually configure each device for a prototype system and store the settings of each device into corresponding DCF file. After storing the settings, they can be used for manufacturing of the systems with equal settings. Further challenges can appear during the assignment of device parameters. Raw configuration files with raw parameter values are typically accessed by software developers, who do not necessarily have a detailed understanding of sensor, drive and system behavior. Control and system designers are not necessarily able to fluently edit the raw files or assign the correct raw values. Typical results have been incorrect or even missing parameter values, because the necessary parameters with their descriptions are described only in text documents. In addition, product specific commissioning tools with dedicated bus access are available for some products [2], which will lead to a need for too many different bus interfaces. Functional safety requirements are tightening up and setting requirements for well defined and documented processes [8] [9]. Main targets are to get managed structural descriptions for the systems, to help with the selection of correct components, and to use them according the specifications and with correctly set parameter values. Following standardized processes results in understandable designs, including design documentation.

```
01 [Tools]
02 Items=2
03 [Tool1]
04 Name=Device Settings
05 Command=encwiz.exe $DCF
06 Wait=1
07 Make=0
 : :
```

Figure 1: Example [Tools] section in an encoder EDS file

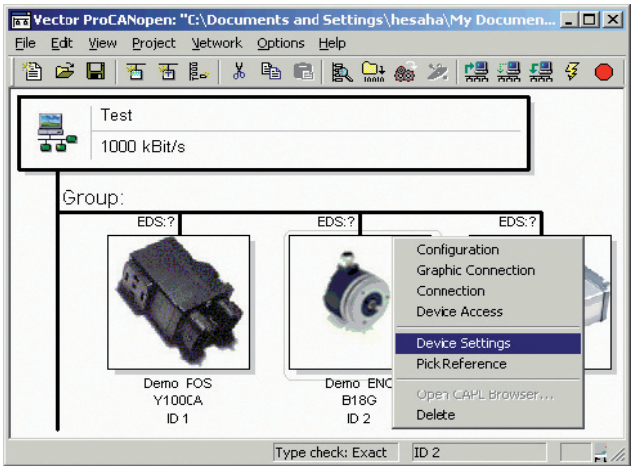

Figure 2: Launching the device editor of the system design tool

CANopen possibilities

Instead of only communication protocols, CANopen specifications include a thorough layered description of communication services, system design process [3] [10], configuration files [10] [11] [12] and integration mechanism between various design tools [11]. CANopen device profiles provide an efficient approach to the re-use of common measurement and drive functions in systems. It has been proved earlier, that using CANopen to coordinate the development of distributed systems significantly improves both quality and efficiency of the software development [3] [5].

CANopen also helps to manage a system's complexity and tolerate differences between components and a sytem's life-cycle. [4]. Following the design process efficiently expects the use of appropriate tools. Multiple system design tools have been on the market for years. ▷

The possibility of linking device specific tools to the system design tools [11] has not become widely known, because product specific commissioning tools [2] have commonly been used. The roles of the tools are clear – system structure and all communications are designed with system design tools and product specific add-ons can be used for adjusting the device behavior.

The clear distinction between the parameter groups improves the process [4]. Device configuration plugins provide a safe access for control and system experts to the device parameters. The approach follows the standardized CANopen design process and moves the focus from editing the raw values in a set of files to adjusting the system behavior. In addition to system design, standard tools can also be used for system troubleshooting, based on system configuration stored as a CANopen project.

The use of a CANopen tool integration interface provides vendor and version independence between the tools. The tools in the example figures are just examples – the presented concepts are not limited to the example tools. Figure 1 presents how links to the external tools can be defined in EDS files, where the links are automatically copied into DCF files during the system design. System design tool passes a path of the corresponding DCF file as a command line option. The command in an example system design tool is highlighted in Figure 2.

Case A: Encoders

The structure of CANopen encoders is clearly defined and thoroughly organized [14]. Main differences are the most common encoder types – linear, single- and multiturn absolute encoders. All types share the same basic structure with measurement direction control, scaling and preset. Advanced features, such as speed computing and CAMs, are not included in the examples of this article, because they are not supported by the majority of the encoders. The type information is available in the higher word of the device type object [14]. Encoder class information is not available and thus supported options need to be determined based on the objects supported by the encoders. Supported warnings and alarms are clearly indicated in the object dictionary, which helps with troubleshooting. Only the support of commissioning diagnostics control cannot be determined unambiguously from the object dictionary.

The measurement unit of an encoder is the most essential information from the system point of view, because it directly affects control software [6]. Encoder configuration wizards can automatically compute the measuring step size for both signal and parameter objects, based on current scaling parameter values. Currently it is possible to determine the output signal unit by combining information from various objects. But it is not possible to assign measurement unit information directly to the signal and parameter objects, because such an entry is not supported by DCF file format. XDC files supporting unit information can be used in the future, e.g. when tool support has been included [12]. The use of XDC files instead of DCF files enables exporting more complete signal specific meta information into the CANopen abstraction layers of application programmable devices [5].

The user interface of the encoder configuration wizard can be made scalable according to the features supported by a device under configuration. Figure 3 presents an example view for a class C1 linear encoder without preset function and with scaling disabled. Measurement direction and scaling selections are presented in human readable form and fields for unused and unsupported parameters are hidden to keep the user interface as simple as possible. The step size has been computed based on the linear encoder measuring step settings.

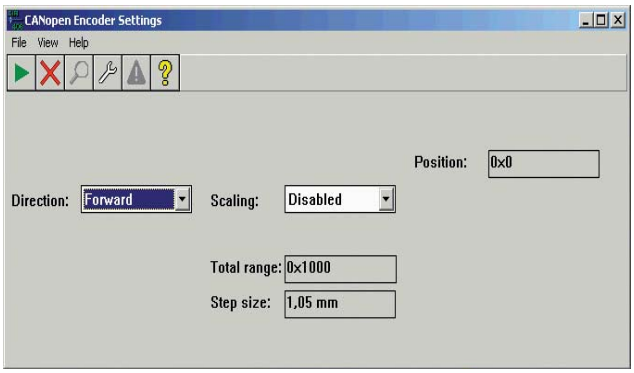

Figure 3: Parameters of a class C1 absolute linear encoder

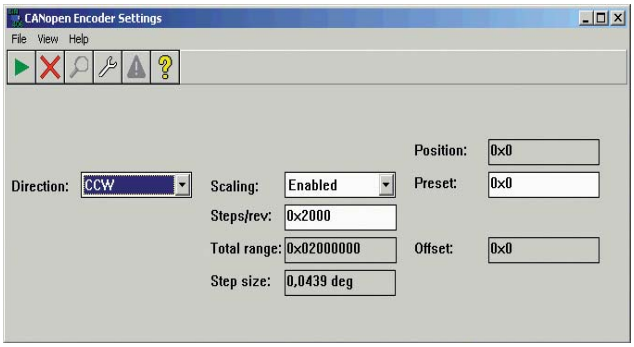

Figure 4: Parameters of a class C2 absolute multiturn encoder

Another example is shown in Figure 4, where a class C2 multiturn absolute encoder with scaling and preset support is under configuration. Direction enumeration has been changed according to the rotary encoders and supported optional fields are included. The step size has been computed based on the assigned single turn resolution and the number of distinguishable revolutions. Preset value is typically adjusted only by the control system. Position and offset fields are targeted mainly for troubleshooting purposes.

Case B: Generic I/O-devices

Generic I/O-devices are the most typical CANopen devices on the market. The device profile is versatile, which introduces special challenges to the general purpose configuration editors. The basics are straightforward – there are dedicated object areas for digital and analog inputs and outputs [13]. There are also well defined optional control blocks for input and output signals organized so that the sub-indexes match with corresponding signal objects. Supported input and output types are indicated in the higher word of the device type.

A problem can arise in the digital inputs and outputs, which are organized as groups of eight signals. With EDS and DCF files it is possible to identify the number of supported digital channels in multiples of eight signals, not one by one [1] [10]. If the minimum and maximum value of a byte object is defined, the lowest and highest supported ▷

bits can be determined, but possibly unsupported bits between cannot be exactly indicated.

Many devices support multiple analog signal types and ranges, which are controlled by manufacturer-specific objects.

The objects can be linked to the standardized signal objects by using object links [10] to enable generic tools to create complete groups of objects for each channel organized to screen in a logical order. The second problem is exposed, when channel specific physical scaling is needed [6]. There are standardized objects for the description of the physical units, but the effect of scaling objects into the units is not completely defined. The device specific parameter editor could provide such services, but it expects the support of corresponding, object specific meta information entry in configuration files. If signal types are selected by physical connections, e.g. by using type specific pins, systematics cannot be provided by CANopen.

Object links can also be used to define feedback inputs of the outputs. The third problem can be recognized if an output can be either digital or analog. Standardized signal objects can be used and they can be assigned together with object links but a method for describing which one is active is missing. Corresponding objects can be included into the same group by object links, but dependencies of the object values cannot be described in EDS and DCF [10]. The same problem also applies to XDD and XDC files [12].

CANopen valve drivers contain all listed characteristics causing problems for a generic configuration wizard. Therefore the example wizard presented in Figure 5 is device specific. The approach enables including device specific details into the configuration wizard. In the example devices, operational mode is a complex parameter, which is divided into three fields – actuator index, control mode and direction. Designers will be confused by such complex raw values without using an editor hiding the complexity. The device specific tool also enables an improved screen layout. The block diagram has been adopted from the corresponding device profile with additional device specific blocks. Despite on the parameter organization in the object dictionary, block specific behavior can be clearly visualized. Albeit the configuration wizard is device specific, it does not increase overall complexity because of the standardized tool integration mechanism.

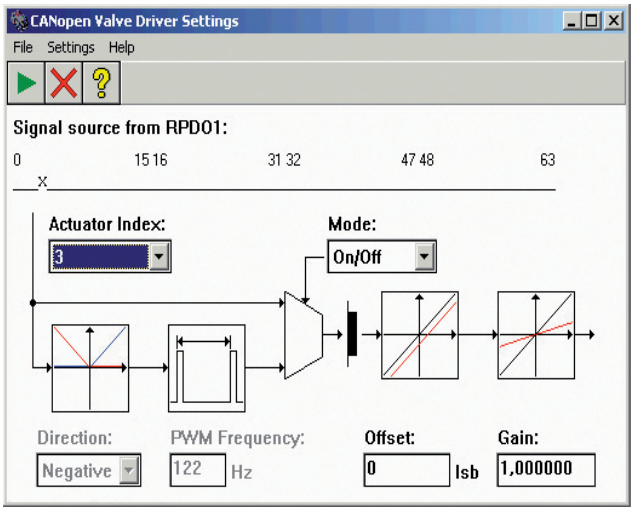

Figure 5: Parameters of a valve driver

Practical experiences

Based on a few years of testing in various projects, the most significant benefit of CANopen device editors is that the main focus can be on the management of system behavior instead of the raw values and file formats. One consequence is that instead of programming experts, control and system experts – who have the best system level knowledge – can take control over the system behavior. Thus, all errors caused by informal parameter modification requests, invalid raw values and violated file formats can be avoided.

Overall performance will be improved in many ways. Setting parameters becomes much faster and less prone to errors, saving time-consuming troubleshooting and correction of the numerous parameter values. Designers need to spend less time reading the device manuals again and again to find out the correct raw values. Especially with simpler devices, knowing the device profile is often enough. Utilization of the CANopen tool integration minimizes the additional effort caused by the configuration wizard. When they have been installed, after passing the acceptance test, extra work is not required. When only one system tool has network access, only one interface adapter and device driver are needed.

The average speed-up of the parameter assignment phase seems to be as high as 60x, even with very simple devices. Absolute time saving in a design of system with 15 relatively simple devices may be over 2 hours. Half of the speed-up comes from tool integration and another half comes from intuitive user interfaces. The most significant result is indirect – configuration errors and inconsistencies, causing dozens of hours of troubleshooting and repair in assembly and service, can be avoided. All improvements apply to each revision cycle of a project.

Future potential

The concepts presented in this article are based on EDS and DCF files, which will be replaced with improved XDD and XDC files. Using the presented concepts with the new file formats makes no sense, until at least tool integration and emergency error code decoding support are added to the file format. Tool integration support is vital, because it enables all information transfers to and from CANopen projects independently of the tools. Emergency error code decoding improves diagnostics efficiency significantly because the use of multiple languages is already supported [12].

Multiple pictures can be linked to each XDD and XDC file [12]. Several figures are needed for each device, e.g. background images of device configuration editors and component figures as icons for system design and configuration download tools.

More detailed meta information is available for signals and parameters [12]. In addition to the minimum, maximum and default values, value enumerations [7], derived types and units [12] are supported. More detailed data type definitions enable the generation of more complete abstraction layer for application programmable devices. Unit and scaling of signal and parameters may be managed based on DCF files [6], but device editors may improve the unit and scaling management by computing the scaling's based in the device profile specific knowledge. Device editors may also use ▷

device specific knowledge, especially for generic I/O devices. It is impractical to include such functionality in generic conversion or code generation tools. Therefore, XDD/XDC format's natively supporting object units should be completed.

Parameter grouping support provides solutions for two problems that are encountered with generic I/O devices. First, boolean signals can be defined as individual signals and mapped into the object dictionary as groups of eight signals. As a consequence, device editors can detect the exact number of supported digital inputs and outputs, and boolean signals can be provided in the generated abstractions as booleans instead of raw bytes [5]. Second, complex parameters can be defined and mapped into objects as groups of signals with complete enumeration, reducing the need for device specific editors.

Parameter grouping works similarly with EDS and DCF files. The main difference is that in XDD and XDC each signal and parameter has its own unique identifier, based on which the linking works [7] [12]. The identifiers can be utilized by device configuration editors for e.g. improving an automatic screen layout management. However, description of parameter dependencies cannot be included without extensions.

References

1] Decker B. M., Improving and testing CiA 401 for the next generation of I/O devices, Proceedings of the 11th iCC, CiA, 2006, pp. 01-01 – 01-05

[2] Lelkes A., Compact drives with CAN interface for industry applications, Proceedings of 13:th iCC, CiA, 2012, pp. 11-08 – 11-13

[3] Saha H., Wikman M., Nylund P., CANopen network design and IEC 61131-3 software design, CAN-Newsletter 3/2009, CiA, 2009, pp. 52–58

[4] Saha H., Benefits of intelligent sensors and actuators throughout the systems life cycle, The Twelfth Scandinavian International Conference on Fluid Power, May 18-20, 2011, Tampere, Finland, ISBN-978-952-15-2517-9, pp. 169–181

[5] Saha H., Improving development efficiency and quality of distributed IEC 61131-3 applications with CANopen system design, Proceedings of 13:th iCC, CiA, 2012, pp. 10-15 – 10-21

[6] Saha H., Unit and scaling management in CANopen projects, CAN-Newsletter 3/1013, CiA, 20013 pp. 8-14

[7] Gedenk T., Use cases and advantages of the new XML device descriptions for CANopen devices, Proceedings of 12:th iCC, CiA, 2008, pp. 06-01 – 06-07

[8] Safety of machinery. Safety-related parts of control systems. Part 1: General principles for design, IEC 13849-1, IEC, 2008, 182 p.

[9] Safety of machinery. Functional safety of safety-related electrical, electronic and programmable electronic control systems, IEC EN 62061, IEC, 2005, 201 p.

[10] CANopen Electronic datasheet – Part 1: General definitions and electronic data sheet specification, CiA-306-1, CiA

[11] CANopen Electronic datasheet – Part 3: Network variable handling and tool integration, CiA-306-3, CiA

[12] CANopen device description – XML schema definition, CiA-311, CiA

[13] CANopen device profile for generic I/O modules, CiA-401, CiA

[14] CANopen device profile for encoders, CiA-406, CiA