CANopen in the frontline of openness

"Open knowledge is any content, information or data that people are free to use, re-use and redistribute – without any legal, social or technological restriction." [8]

Author

Dr. Heikki Saha
Scientific coordinator
CANopen Competence Center Finland
University of Vaasa
Wolffintie 34
FI-65200 Vaasa
Tel.: +358-29-449-8000

Link
www.uva.fi

Open data or open knowledge is one of the most recent IT megatrends [8]. All around the world, data collected by public funding has already been opened to free use and more data is made public all the time. The main problem has been that charging for the use of information collected or produced by public funding has reduced the use of data and related business potential.

All such public data is typically stored in a poorly structured and inaccessible format, often as printable documents, understandable only for human. Opening data means publishing open data in a format that understandable for machines, not changing the status of the data. Quite often usable data is presented in a format that makes data access impractical. Furthermore, individual use of such data may not make sense or provide any big benefits, but combining open data from different sources may provide unexpected benefits.

Control systems are not an exception. There is lots of data freely available and commonly used for various purposes. Data was frequently copied manually and during the design process, which led to a heavy maintenance workload, the use of outdated data, and serious inconsistencies, even between products from the same company. The main reason for this is the fact that data contents are related to service and maintenance and not directly related to the main functions of systems. There are also indirect effects, causing additional delays and workloads with troubleshooting and spare part changes.

A lot of public data is used in CANopen-based distributed control systems. Typically small subsets of data were manually inserted into tools and control applications, which are updated only after serious problems. Another problem is that such data is often hardcoded into applications, so that each update leads to re-testing and re-certification. Updates are in many cases essential, because e.g. new device profiles and SDO abort codes are introduced during development of various standards. New device vendors emerge onto the market, too. Information content often looks constrained and simple, but is continuously updated. Thus, standardized information, storage format, and automatic updates can improve the maintainability of control applications, assembly lines, and service tools.

Table 1: Generic layout of currently available datasets

Dataset	Column 0	Column 1	Column 2	Column 3
CANopen device and application profiles	CiA profile number	Profile title	-	-
CANopen Vendor-IDs	Vendor-ID	Company name	Department	-
SDO abort codes	Abort code [1] [2]	Description en-us	Description fi	Description sv-se
SI-unit definitions	Unit name	Unit abbreviation	Unit code [3]	Specification number

Open documents

CANopen has always been open by nature. All release status standard documents are freely available for all interested parties, including all draft standard documents, except those which are international or European standards and thus distributed by standardization organizations. Draft standard proposal documents are primarily available for members only, but published for all interested parties as excerpts, instead of full documents. Open standards make CANopen attractive and have increased the size of the whole ecosystem, when compared with competing integration frameworks which are not open and free. Open and free availability guarantees that even small enterprises can bring innovative products into the market with lower costs, unlike by using e.g. J1939, Devicenet, and Profibus.

Common terms and acronyms with their official explanations are available as a freely available document, the CANdictionary. It provides a good starting point for newcomers to learn the basics and practical cross references for experienced people. The original one was written in English but Chinese, Finnish and Russian ▷

Table 2: Open datasets currently available from CiA website in machine understandable format

Dataset	URL
CANopen device and application profiles	http://www.can-cia.org/fileadmin/cia/files/cia_profiles.csv
CANopen Vendor-IDs	http://www.can-cia.org/fileadmin/cia/files/Vendor_ID.csv
SDO abort codes	http://www.can-cia.org/fileadmin/cia/files/sdo_aborts_codes.csv
SI-unit definitions	http://www.can-cia.org/fileadmin/cia/files/si_units.csv

translations are also freely available. The CAN Newsletter is a magazine published by CiA, which covers the most recent applications and research results in the area of CAN, CANopen, and other CAN-based system integration frameworks and application layer protocols. A selection of older magazine articles is available on request and from 2012 onwards complete editions can be found online. CiA has organized international CAN-conferences s
provides ope
some of the conference papers since 1995 and all papers since 2000.

Open design information management

CANopen also defines file formats for describing product identities, device type specific interfaces, and application specific configuration information. Standard file formats enable unambiguous design information process. Standardized information transfers enable efficient utilization of larger ecosystem, e.g. in component supply and subcontracting of both hardware and software components of various sizes. The use of a standardized information transfer also enables tool independence, enabling possibilities to improve the information management tool chain. Furthermore, subcontracting can be easily managed, when all involved companies follow the design information storage standards.

CANopen profile database (CPD) files enable providing application- or SW-component-specific interface management and methods for describing validation criteria for electronic datasheet (EDS) files [5]. Device configuration files (DCF) may also be validated by themselves, because in addition to the current parameter values, they contain all the information that EDS files contain [4]. The name of an EDS file, from which the DCF has been generated, is included in the file. EDS files enable systematic documentation of product capabilities, reducing a need for literal documentation [4] [9]. The number of discussions between suppliers and system integrators is also reduced, because the majority of the essential information is comprehensively described. Standardized device data enables efficient computer-aided designs with minimal manual work, ▷

```
01  import urllib2, os, sys                  # Import required modules
02
03  def saveUrl(srcUrl):
04      # Prepare source URL and path for target file ---------------------------
05      (exeFold, exeName) = os.path.split(sys.argv[0]) # Get local folder
06      (rootUrl, fileName) = os.path.split(srcUrl)     # Get file name from URL
07      tgtPath = os.path.join(exeFold, fileName)       # Build target path
08      # Store URL into target path --------------------------------------------
09      httpResp = urllib2.urlopen(srcUrl)  # Get file from given URL
10      output = open(tgtPath,'wb')         # Open target file
11      output.write(httpResp.read())       # Write into target file
12      output.close()                      # Close the file
13      return tgtPath                      # Return target file path
```

Figure 1: Example code for getting a dataset from given URL and storing it into a local file

increasing the overall efficiency significantly.

There are EDS file editors freely available from different vendors to help create the correct EDS files. EDS file checker is available for free to enable device suppliers to create error free EDS files for their devices. Thanks to those tools, inconsistent EDS files have not been a noticeable problem in recent years. An official conformance test program also verifies the completeness and correctness of EDS files.

Information transfer from design into assembly and service is important. Any erroneous or outdated document may introduce significant failure costs in the assembly line and service. In addition to delayed operation, documentation failures may introduce additional component consumption and additional time needed for disassembling assembled faulty components. Therefore, the use of standard DCF files as a transfer format from design to assembly and service [4] provides a significant increase in quality and performance.

In addition to tool independence, EDS and DCF files provide an integrated mechanism for integration of various design tools [6]. Such a mechanism enables the flexible utilization of generic and target device specific tools without increasing the required workload during the design process.

Open constant data

Certain constant data is needed for various purposes during system development. Device profiles, vendor-IDs and SDO abort codes are needed in design, assembly and service tools as well as in the control system GUI devices. SI-units are mostly needed for signal scaling in the application development and for signal scaling and visualization in GUI devices. Analysis tools get unit information from communication databases, where the unit information currently needs to be added manually. Unit information is not directly supported by EDS and DCF files, but in case of many device profiles, unit information may be combined from values of other objects [10].

Traditionally constant data is manually inserted separately in each project, which results in inconsistent information contents and an extensive decrease in both quality and efficiency. While adding the information seems to be simple, one should remember that all information is continuously updated and updates should be available for each purpose with a minimum effort.

CANopen data is available in a comma-separated values (CSV) format, where a semicolon is used as a column separator. The use of columns is included in Table 1. Another format may not provide significant benefits, because the data currently describes certain enumerations only. The first row always contains the column titles and further rows contain the actual data. Additional translations may be provided for SDO abort codes and are added as further columns. The same language coding scheme, which is used in XDD and XDC files, is used for column titles of abort code descriptions [7].

In addition to the SI-unit descriptions, unit prefixes have been defined [3]. They don't need to be published as a table, because prefix factor values are defined as 10[prefix value] and literal prefixes are the same, independent of the language.

Getting open constant data

Currently supported datasets and locations are listed in Table 1. Vendor-IDs and supported profiles have been provided by CAN in Automation (CiA) since 2008. SDO abort codes and SI-unit descriptions have been collected and provided with Finnish and Swedish translations by the CANopen Competence Center Finland. Further information was published to enable application developers to keep the generic constant data up-to-date, without suffering from repetitive manual work.

Two generic sample code examples are provided to give everybody an easy start for getting the data. Python was selected as an example language, because it is a productive language and available for free. For more details, readers are advised to refer to the Python documentation. The first example is the function saveUrl in Figure 1, which takes an URL of a dataset as an argument, loads the given URL into a local file and returns the full path of the file. By default, in lines 4 to 7, the target path is determined to have the original filename from the given URL and same folder with the loading code module. Lines 9 to 12 load the given URL and store the contents into a file with a previously generated path.

After getting the local copies of the data, it should be translated into an usable format. The example function ▷

```
01  import csv      # Import CSV module
02
03  def getCsv(srcPth):
04      d = {}      # Prepare result dictionary
05      flds = []   # Prepare field array
06      # Parse column names ----------------------------------------------------
07      dictReader = csv.DictReader(open(srcPth, 'rbU'), fieldnames = flds,
08          delimiter = ';', quoting=csv.QUOTE_NONE) # Read CSV
09      hdrs = dictReader.next()        # Read first row
10      for dd in hdrs:                 # Get result
11          nrOfCol = len(hdrs[dd])     # Get number of columns
12          flds = hdrs[dd]             # Get column names
13          for i in range(0, nrOfCol): # Loop through columns
14              d[hdrs[dd][i]] = []     # Use column items for dict keys
15      # Parse column data -----------------------------------------------------
16      dictReader = csv.DictReader(open(srcPth, 'rbU'), fieldnames = flds,
17          delimiter = ';', quoting=csv.QUOTE_NONE) # Read CSV
18      hdrs = dictReader.next()        # Read first row to move read point
19      for row in dictReader:          # Read rows
20          for key in row:
21              if key in flds:         # Take only existing fields
22                  d[key].append(row[key])
23      return d                        # Return as dictionary
```

Figure 2: Example code to read a CSV-format dataset into a dictionary

getCsv in Figure 2 reads any CSV format file by utilizing a standard csv library and returns the contents as a Python dictionary. The CSV file is read and parsed into rows and columns in lines 7 to 8. The first line is picked at line 9 and column names are extracted into an array in lines 11 to 14. Source data may consist of any number of columns and the column names are read from the file itself. Therefore the actual values need to be read in another phase, in lines 16 to 17 by using previously read column names as dictionary keys. The read pointer is moved into the second row in line 18 in order to get only the actual data from the file. Lines 19 to 22 read the data into the dictionary. The resulting dictionary contains columns named according to the first line of the CSV file. There are array items for each row.

Summary

Common, public constant data has been available since the beginning of CANopen, but first in a format only readable by humans. The main problem has been that the data is continuously updated. Especially vendor-IDs but also device profiles and sometimes new abort codes and units have been added. Automated updates are saving time and effort required for updates and removing typing and translation errors.

A uniform presentation of data can be achieved by adopting an automated use of open data. As a consequence, system operators and service personnel will not be confused by differently written vendor names, SDO abort codes with inconsistent wordings, differently named device profiles or units. Design engineers need not spend several hours for regular updates of such data. Thus, all people involved can focus on more productive work.

Future plans include the opening of at least error register flags, object dictionary command keywords, NMT-states and -commands in order to provide uniform wordings in several languages. Currently, information contents of e.g. error register flags to the service personnel have been limited, because of the non-uniform wording. Furthermore, official short descriptions are not currently available for object dictionary command keywords.

At the time of writing, Ixxat and TKE offer products supporting automatic updates directly from CiA website. If there are others, those companies are invited to provide information, for example in the CANopen Linked-In group. Readers are also invited to provide proposals for new datasets to be opened and possible improvements for currentdatasets.Let'simprove the CANopen world together! ◀

References

[1] CANopen application layer and communication profile, CiA-301, CiA
[2] Representation of SI units and prefixes, CiA-303-2, CiA
[3] Additional application layer functions – Part 7: Multi-level networking, CiA-302-7, CiA
[4] Electronic device description – Part 1: Electronic Data Sheet and Device Configuration File, CiA-306-1, CiA
[5] Electronic device description – Part 2: Profile database specification, CiA-306-2, CiA
[6] Electronic device description – Part 3: Network variable handling and tool integration, CiA-306-3, CiA
[7] CANopen device description, XML schema definition, CiA-311, CiA
[8] Open knowledge foundation, https://okfn.org
[9] Saha H., Experimental CANopen EEC management, CAN-Newsletter 1/2013, CiA, 2013, pp. 12 – 18
[10] Saha H., SI-Unit and scaling management in CANopen, CAN-Newsletter 3/2013, CiA, 2013, pp. 30 – 34