

# CANopen Master Source Code

## Embedded Software for CANopen

The Master Source Code allows the user to expand his products' firmware with CANopen master functionality. The software is especially suited for integration into control systems.

### Features and Advantages

The Master Source Code makes available all functions necessary for a CANopen master. This implements the following standards of the user organization CAN in Automation (CiA):

- > CiA DS 301 Version 4.02, Application Layer and Communication Profile
- > CiA DS 302 Version 3.1, Framework for programmable CANopen Devices
- > DSP 405 Version 2.0, Interface and Device Profile for IEC61131-3 Programmable Devices

Because of its generic structure, slaves with any profile can be connected. The user can choose to incorporate the software into a project or construct the CANopen functionality as an independent task in the system.

With the use of the CANopen source code, significant time savings can be achieved during product development. The user can concentrate on the integration of his own application; the implementation of the CANopen protocol is thus simplified significantly.

### Functions

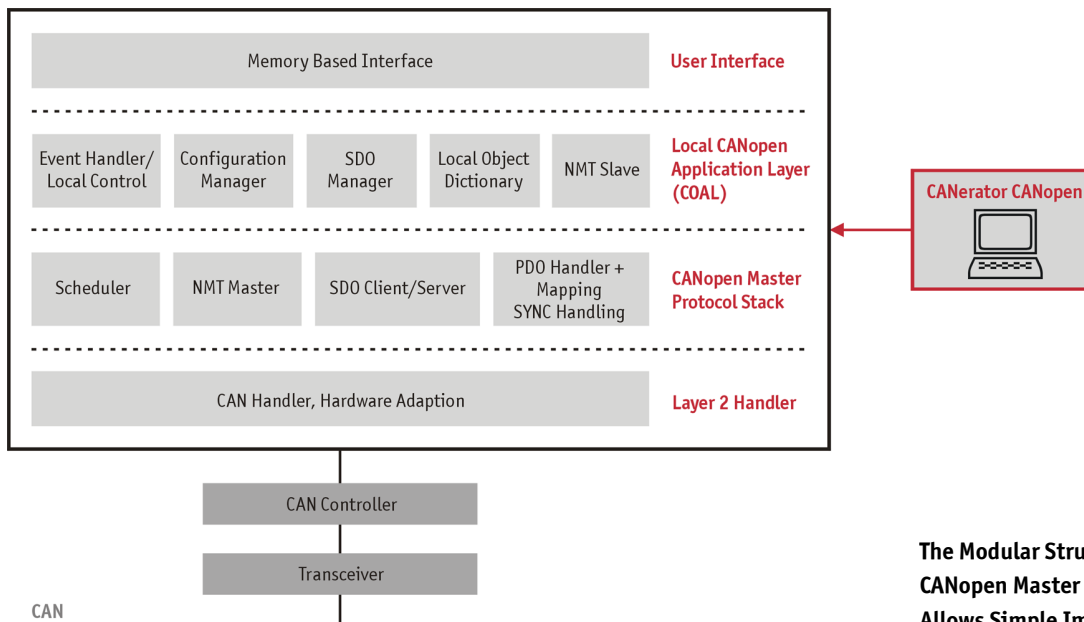
The source code offers the following functionalities:

- > Network Management (NMT) – control of the connected nodes via the NMT message. Guarding and heartbeat are still carried out.
- > PDO Handling – the actual process data are transmitted via PDOs. The PDOs can be configured via the object directory. PDOs are also handled synchronously.
- > Local Object Directory – the code makes available its own object directory via the bus, which can be expanded by the user.
- > SDO Handling – the user can initiate individual SDO requests via the SDO handler.
- > Configuration Manager (CMT) – with this, central configuration of the connected CANopen nodes can occur in embedded systems.

### Special Functions

The software supports the storage of data in non-volatile memory. There are interfaces available for connection to a file system or flash memory.

On system start, the "boot slave" functionality is supported completely. The network management itself takes care of the correct starting and configuration of the connected nodes. The failure control is also taken over by the CANopen master.



**The Modular Structure of the CANopen Master Source Code Allows Simple Implementation.**

V1.4 2005-7

The CANopen Master Source Code from Vector is available for many popular microcontrollers. For current information, please visit our homepage on the Internet at:

[www.vector-informatik.com/sourcecode](http://www.vector-informatik.com/sourcecode)

### Application Areas

The code can be used everywhere customers would like to equip their control systems with CANopen functionality. Users are first and foremost companies that manufacture CANopen devices, as well as system integrators who must create entire systems.

### Hardware Interfaces

The software is written in ANSI-C and therefore can run on the most various platforms. At a minimum, the use of a 16 bit microcontroller should be considered. For addressing various CAN controller types, adaptations are available. The interface of the CAN driver is designed so that it is an easy matter to move to other CAN controllers.

Board Support Packages (BSP) are available for a few evaluation environments. A BSP consists of a makefile, a text file with the settings of the #defines, and a file with the necessary system adaptations. Even if the user has another target environment, these files provide valuable information on porting to customer hardware. Currently, the following BSPs are available:

- > Adaptation for evaluation board ec376 with Motorola 68376 from Würz Elektronik GmbH and GNU compiler or Microtec Research Compiler
- > Adaptation for kitCON 164CI with Infineon SAB164CI from Phytex GmbH and tasking compiler

- > Adaptation for Vector CANopen driver under Windows 95/98/NT and Microsoft compiler VC++ 6.0
- > Power PC

### Data Interface

The data interface makes it possible to uncouple the application from the CANopen stack as the communication task. This makes it possible to implement a single processor or a multi-processor system. In the latter case, the data interface is placed on a physical DPRAM:

- > Command queue – the command queue is used by the control system to send requests to the application layer. This includes transmitting initialization data or requests of function components (for example to execute SDOs).
- > Event queue – the event queue carries the data stream from the application layer to the control system. This data consists essentially of asynchronous confirmations (for example a response to an SDO request) and events (for example error states).
- > Process image – the application acts with the CANopen network essentially by means of a process map. This map contains all I/O data and other process variables that are to be transmitted by the CANopen master.

### **Network Management**

This module assumes responsibility for administering and monitoring nodes that are connected in the network. For this to happen, the nodes that are to be accessed and monitored must be declared. The corresponding parameters for error monitoring (guarding and heartbeat) must also be transmitted. This information is stored in the local object directory and can then be requested externally via the bus and internally via the DPRAM. The monitoring and start of nodes runs largely automatically.

### **PDO Handling**

The PDO handler has the task of accepting received PDOs from the slaves and forwarding them to the process map of the data interface. In the other direction, it transmits values from the application to the slaves. This can take place cyclically (by SYNC) or it can be event-driven (by change).

The assignment between PDO and storage of data takes place exclusively via the local object directory. The application is thus completely uncoupled from concrete communication tasks.

### **SDO Handling**

The SDO handler makes it possible for the application to have access to the object directories of the connected CANopen nodes. It is possible to transmit data blocks in either direction. The "expedited download/upload protocol" is used for objects up to a size of 4 bytes. For objects larger than 4 bytes, the "segmented download/upload protocol" is used.

The block transfer is also implemented and can be initiated in a number of ways, one of which is by the user.

It is possible to run multiple SDO transfers in parallel (only for different SDOs).

### **Local Object Directory**

The Local Object Directory enables access to the master functions and to master parameters and variables over the network. This makes it possible to configure the master over the network or to operate it through locally available functions (displays or inputs and outputs). The Configuration Manager is especially activated this way.

The variables in the object directory can be accessed by the application through the corresponding SDO read/write requests by indicating the own node ID.

### Configuration Manager (CMT)

The Configuration Manager has the task of saving parameter sets of the CANopen slaves and distributing them at a certain time to the connected nodes on the network. The time of this distribution can be determined in the following manner:

- > When the master is started, a configuration is performed for all nodes that have been entered in the list of assigned slaves and for which a configuration is available.
- > If guarding or heartbeat fails, the configuration is initiated again.
- > Initiating the configuration of a specific node at any time by the application – for example in order to subsequently configure connected sub-networks.

In this context, the configuration of CANopen devices refers to setting parameters in their object directory. The parameters in question are mainly PDO parameters and mapping information.

### Support during Integration

Naturally we can also support you during the integration of the CANopen Master Source Code into your environment. Our support services range from custom-tailored training solutions to workshops to project work.