

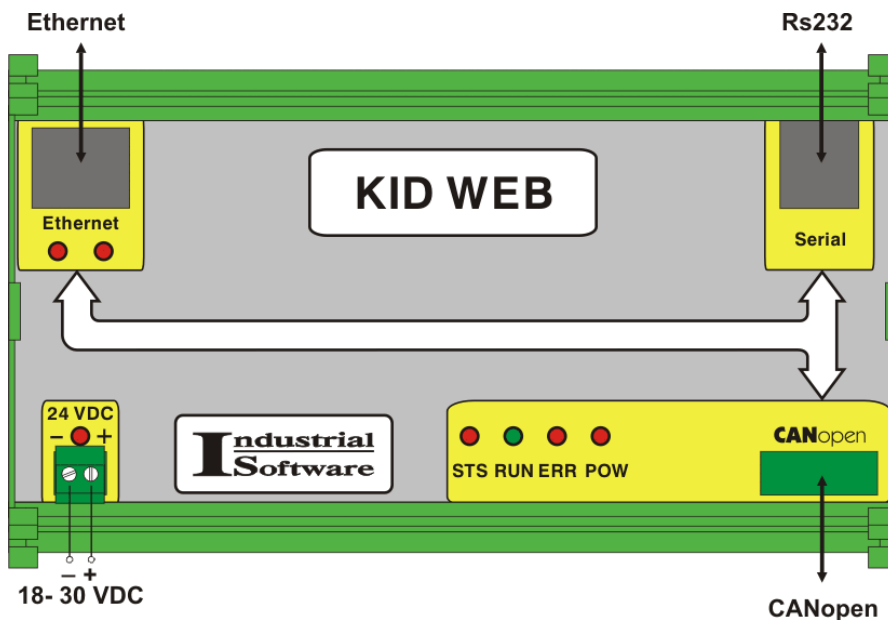
# KIDWeb rev. 2

## ( CANOpen Ethernet Gateway) Technical Datasheet

*Revision 1*  
*Date: 04.05.2007*

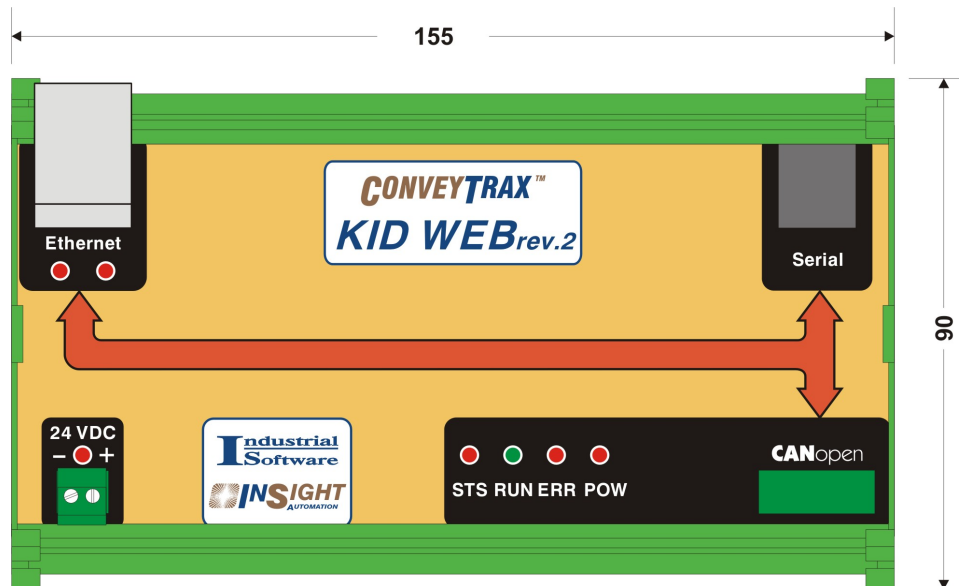
### General Information

The KIDWeb module is a CANOpen to Ethernet intelligent bridge. It has been implemented as a CANOpen master, therefore using COB-ID 0. From the Ethernet perspective, the KIDWeb is a simple host, whose interfaces and IP services can be used for controlling and monitoring the CAN network that is connected to it. Ethernet is the main interface because it can control the behavior of the KIDWeb module, its configuration, and serves as the entry to the real bridge functions. The KIDWeb cannot be configured through its CANOpen interface. There are three main functions of the KIDWeb module. The first is to provide the means for reading and writing specific objects from the modules' object dictionaries. This is done through a few telnet commands. The second important function is to be able to upgrade the firmware of the CANOpen modules residing on its CAN network. This is accomplished through both telnet and FTP actions. The third important function is to provide a sort of CAN sniffer that when configured send broadcast/unicast UDP messages holding the specified CAN messages whenever the later appear on its CAN interface.



## Physical Dimensions [mm]

- Dimension: 155mm x 90mm x 45mm



## Electrical Specifications

- Power requirement:
  - Nominal: +24VDC/ 80mA
  - Voltage Range: 12V to 30VDC
- Operating temperature: -20°C ~ +80°C
- Storage temperature : -40°C ~ +85°C
- Protection – IP 20
- Isolated voltage:
  - DC/DC converter from power supply – 1000VDC
- Mounting: DIN Rail
- Electromagnetic compatibility: IEC 61131-2

## Communication Interfaces

RS-232 Serial Port – Used only for debugging and low level firmware upgrades

CAN – provides CANOpen functionality as a master on the CAN network

Ethernet – provides services through the following protocols:

Telnet – used primarily for configuring the module and issuing commands that read/write objects from the CANOpen network, or for sending messages on this network

FTP – usually it is used for storing firmware and configuration files for later use.

UDP – mostly used for monitoring CANOpen messages, but also providing the basis for the algorithm that changes the module's IP address.

## Basic Functions

### CANOpen Network Mode

The KIDWeb, being a Network Management master ( NMT Master) has the ability to Start/Stop/Restart the CAN network. The user can access those functionalities by using the “ms”, “mt”, and “mr” telnet commands. In addition to this, the KIDWeb has the configuration option of automatically starting every module that boots up. This last option is controlled by

the “mb” command. For a thorough explanation of how those commands work, please consult the “KIDWeb Command Reference”

### **CANOpen modules monitoring**

The KIDWeb module sends broadcast packets containing status information for all Industrial Software devices on its CAN network. Those packets are UDP datagrams and each packet contains a field for every possible module. The packets are sent at a specified interval or whenever some module’s status changes. The KIDWeb uses the Node-Guarding protocol to keep track of the available modules. For more information on how to stop this behavior or how to change the interval, please look at the “mm” command and its associated appendix in the “KIDWeb Command Reference”.

An interactive module status monitoring is available through the “s” telnet command. As a response, the KIDWeb sends a list of all the modules it has found on the CAN network and their current state: operational/pre-operational/etc...

### **CANOpen Object Dictionary operations**

The most basic feature of the KIDWeb is that it provides an interface for accessing the object dictionary of the underlying CAN modules. The KIDWeb can read/write 1 to 4 bytes, or it can read a whole domain object. For those operations, the user needs to supply the node-id of the module, the object index, sub-index and the length of the data if writing data. Dictionary read ( “dr” command) simply needs the node-id, the index, and the sub-index. It reads the specified data and displays it regardless of its size. Reading domain objects is the same as reading normal objects. The KIDWeb will display the data regardless of its length. When writing however, the user must be aware of how big the target field is ( 1, 2, or 4 bytes) and supply that argument together with the data itself. Writing a domain object is used only for firmware upgrades of the CAN modules. For more specific information, read the explanation of the “dr”, “dw”, and “du” commands in the “KIDWeb Command Reference”

## **Advanced Functions**

### **Node-guarding**

In order for the KIDWeb to know what modules are on the network, it uses two algorithms. The first one is passive: The module monitors all the BOOT messages sent by modules that have just been powered up and notes their existence. The second algorithm is active: the KIDWeb consecutively sends a query (every 100ms) to every possible node-id and waits for a response. If a module responds, it is noted as available and its state is recorded in an internal table. Node-guarding generates constant CAN traffic to and from the master node, so in some situations it can cause trouble. One of those situations is if you have 2 KIDWeb modules on the same CAN network. Although this is not allowed by the CANOpen specification, it can be beneficial in some situations where you need one KIDWeb for controlling the network and another for pure monitoring. In this situation, the node-guarding function of one of them needs to be disabled. Usually this is done on the monitoring KIDWeb, as its sole purpose is sniffing the CAN network and doesn’t take part in controlling it. Enabling and disabling the node-guarding function is accomplished through the “mg” telnet command.

### **CANOpen message monitoring**

The KIDWeb can be configured to monitor certain CANOpen messages. Namely, those are the PDOs and their COB-IDs range from 0x180 to 0x580. Then it is configured like this, whenever a PDO is encountered, the KIDWeb stores it in an internal buffer and then sends a broadcast/unicast UDP message on the Ethernet. Since UDP is a connectionless protocol, those UDP datagrams may never reach the host that expects them, especially in a

high network load situation. Therefore, every UDP datagram is transmitted 3 times. This however may mislead the monitoring application that there were 3 CANOpen messages that triggered those 3 UDP messages. To solve this, the KIDWeb keeps an internal counter that also gets sent in the UP message. This counter's purpose is to differentiate consecutive messages from their retransmissions. Whenever a UDP packet is sent and retransmitted, the KIDWeb increments this counter. The application that monitors those UDP messages should monitor this counter field in the packets and discard the retransmissions. Let's suppose the application receives a UDP packet whose counter is 5. It reads the rest of the data and handles it. The next packet received also has a counter value of 5, so the application knows that it has already handled this packet and discards it. The next packet has a counter of 6, so the application should note this counter change and handle the data. If another packet with counter 6 arrives it should again be discarded.

This "CANOpen sniffer" usually uses broadcast UDP messages, thus allowing multiple recipients of its data. This however increases network demands. If this broadcast functionality is not needed, it can be suppressed by the "gb" telnet command. If suppressed, the KIDWeb will send UDP messages to the PC that was the last to register a PDO for monitoring. Registering a PDO for monitoring is accomplished by supplying its COB-ID to the "gas" or "gam" telnet commands.

Emergency messages are always monitored. When one or more emergency messages are received, they are transmitted as UDP broadcast messages. Since emergency messages are generated in well... *emergency* situations, this monitoring cannot be switched off. The UDP datagrams holding the ER messages are also retransmitted to minimize the chance of some host missing them.

### **Inter KIDWeb communication**

A progression of the "CANOpen message monitoring" is a feature that lets you go past the 127 node limit of the CAN network. Just like the message monitoring algorithm, this one listens for specific PDOs on the CAN network and stores them in an internal buffer. This buffer is transmitted as a UDP message to a different KIDWeb. The other KIDWeb receives the UDP message, parses the data and if configured sends a PDO in its own CAN network. This mechanism allows a module from one network to send a PDO to a module in another CAN network. This scenario is controlled by a special mapping file called CONN.BIN. It holds quartets of 2 IP addresses and 2 COB-IDs arranged as a source pair and destination pair. This file is prepared by a high-level application and then downloaded to all the KIDWeb modules that take part in this logical KIDWeb network. Each KIDWeb parses the file for entries that specify it being either a source or destination and creates a map for this.

Let's say that a CONN.BIN entry holds the following information: Source COB-ID – 0x200, source IP – 192.168.0.10, destination COB-ID – 0x300, destination IP – 192.168.0.20. Now, let's suppose that some module in the CAN network managed by 192.168.0.10 sends a PDO with COB-ID 0x200. The KIDWeb receives it, searches through its table and realizes that it is configured as a source, so it sends a UDP message out. This UDP message finds its way through the Ethernet and reaches the KIDWeb with IP 192.168.0.20. It has already parsed its CONN.BIN and knows it is a target, so it scans through the message and finds the PDO. It knows however that it should transmit this message in its own network with a different COB-ID, namely 0x300, so it takes that data from the original PDO and transmits a new PDO with COB-ID 0x300 and the original data from the 0x200 PDO. In a properly configured example situation, there will be a module in the second CAN network that receives the 0x300 PDO and handles the data.

This is a very powerful algorithm that has to be used with great care as the KID Webs involved serve a very specific function and rely on the high-level software to create the proper

mapping. Therefore the entire burden is on this high-level software to properly distribute the PDOs across the CAN networks and to properly configure the modules to send/receive specific PDOs. For an in-depth explanation of how the CONN.BIN file is organized and a diagram of a different example, please read the appendixes in the “KIDWeb Command Reference”

### **CANOpen module firmware upgrades**

Firmware upgrades for the KIDWeb and the modules in the CAN network are relatively the same procedures. Both start with uploading the firmware file to the KIDWeb through FTP. The file is stored in the internal file system and is accessed later on by the “u” command. Please note that the upgrade procedure is specific to Industrial Software’s devices and should only be used with them. We cannot guarantee that the algorithm will be useful for upgrading generic CANOpen devices. Some devices don’t offer such functionality and some stick to the specifications and may be upgradeable. When upgrading the KIDWeb’s firmware, if the file is found and the name suggests a proper firmware file, telnet will return “O.K” and the connection will be suspended. Since the module reboots after firmware upgrade, it’s the responsibility of the client to close the connection, as it will not be a valid connection once the module reboots. When upgrading modules on the CANOpen network, the connection stays open and when the “u” command finishes upgrading, the connection can be used for further issuing of commands. The upgrading of modules is actually a long domain download. The first message, the initiating SDO will force the module to erase its flash, therefore rendering it unusable as a specific module. Its kernel however will remain functional, so even if the firmware upgrade is not successful, the user will be able to access the module and retry the upgrade.

### **Automated CANOpen module upgrade/config**

This mechanism is intended for fully commissioned systems that will not be reconfigured soon. It involves 2 files, a firmware file for the CAN modules and a special configuration file being uploaded to the KIDWeb. Whenever a module malfunctions, personnel can replace it with a new module with the same node-id and baud settings and then hold the install button until all the LEDs on the module light up green. This signals that the module has entered and auto config/upgrade state. The module will then start sending emergency messages to the KIDWeb. Those messages, when heard will get acknowledged and the LEDs will become red. At that point, the KIDWeb searches its file system for a configuration file and a firmware file, checks the new module’s firmware revision and if needed upgrades the module with the firmware found on its file system. After the upgrade process is complete, the KIDWeb parses the configuration file and according to it does a mapping configuration of the module, thus restoring the configuration found in the old malfunctioning module. The result is a 2 step replacement procedure for a bad module. If two or more modules need to undergo this procedure, they will be handled one after the other. After the first module starts upgrading, the KIDWeb will not acknowledge any other upgrade/config requests until it has finished with the first module, then for the second and so on modules will stay with their LEDs green until the KIDWeb finishes its job with the current module.

### **Sending user-defined CAN messages**

The KIDWeb module supports user defined messages to be sent to the CANOpen network. The user may use the “tm” command to send any a 1 to 8 byte long data using COB-IDs 1 to 0x7FF.

### **Specialized CANOpen state monitoring with Nonvolatile RAM.**

This feature allows data from certain PDOs to be copied in an internal table that has batter-backed up power supply. The default behavior is OFF, so nothing is copied to the Nonvolatile memory. If the KIDWeb is configured as STAGING\_MONITOR, it will receive those special PDOs and store their data in NV memory. This data can be read through the “wrm” and “wrz” telnet commands. With this feature, the modules can be configured in such a manner, that they send vital data to the KIDWeb that will be preserved even in a power-loss situation. The KIDWeb only stores the data and is not responsible for restoring it to the modules upon power-on.

The KIDWeb can also be configured as STAGING\_CONTROLLER. In this situation, the KIDWeb will acknowledge the received data, by sending a CAN message with the same COB-ID it just received and only the first byte of the received data. In situations like that, the modules sending the data should be configured in such a way that they will wait for the response. After the response is received, the module can be sure that the KIDWeb has accepted the vital data and has stored it in ne NV memory.

Scenarios like this are used in the staging modules. The KIDWeb keeps track of all the modules states, what trays they hold and what actions every module wants to take. In this example, the CAN modules send a 6 byte request. The six bytes hold state information and the tracking numbers of the trays the module is currently handling. Every time one of the modules wants to change its internal state ( for example wants to accept a tray from the upstream module) its sends a request to the KIDWeb with the state it wants to go to and the tracking info associated. The module then waits. If it doesn't receive an acknowledgement message, it will not proceed. Only after the KIDWeb acknowledges the data, the module starts its motors and initiates the current actions. This procedure ensures that if at any moment, the power goes off, the KIDWeb will be holding a complete picture of the whole network and the states of all the modules. This information can be used later, to restore this information to the modules. Once restored, the modules can be ordered to proceed and operation will resume as if there was no power loss.

This algorithm involves specialized module firmware. It's also very important to note that the restoration of data to the modules is done by higher level PC software. The KIDWeb only holds that data, it doesn't interpret it in any way. The PDOs used for this algorithm are with COB-IDs 0x201 to 0x2BF, excluding 0x240 and 0x280. The theory behind those numbers is as follows. Every staging module is logically divided into 3 zones. The maximum number of staging modules on a single network is 64. Every zone has a PDO allocated for communication with the KIDWeb. The COB-ID for a particular zone is determined by the node-id of the module and the type of the zone ( center/left/right).

Center zones use a COB-ID equal to  $0x200 + \text{node-id}$ .

Left zones use a COB-ID equal to  $0x240 + \text{node-id}$ .

Right zones use a COB-ID equal to  $0x280 + \text{node-id}$ .

Since we always add the node-id, and node-id of 0 is not allowed, the KIDWeb will not monitor 0x200, 0x240, and 0x280. If the user needs to access the data sent by a particular module, he can use the “wrm” telnet command with the node-id of the module. One can also read only the data for a particular zone, but for this, he should send the zone-id which is not very logical. Zone ids are not node-ids. They are the same as the COB-ID that corresponds to this zone, but without the 0x200. so if we need the left zone on node-id 5, we can access it with the “wrz” command and 45 as an argument. The right zone on the same node-id will be addressed with 85.