

Yacht Devices

User Manual

Yacht Devices NMEA 2000 Bridge YDNB-07
also covers models
YDNB-07N, YDNB-07R

Firmware version
1.07

2018

© 2018 Yacht Devices Ltd. Document YDNB07-003. April 23, 2018. Web: <http://www.yachtd.com/>

NMEA 2000® is a registered trademark of the National Marine Electronics Association. SeaTalk NG is a registered trademark of Raymarine UK Limited. Garmin® is a registered trademark of Garmin Ltd.

Contents

Introduction	4
Warranty and Technical Support	6
I. Product Specification	7
II. Installation and Connection to NMEA 2000 Networks	9
III. LED Signals	11
IV. MicroSD Slot and Card's Compatibility	12
V. Loading of Programs into the Device	14
VI. Structure and Basic Syntax of the Program	15
VII. Description of the Settings	17
VIII. Software Filters	21
IX. Optimization and Performance	30
X. Debugging of the Program	34
XI. Firmware Updates	36
Index	38
Appendix A. Troubleshooting	39
Appendix B. List of NMEA 2000 Messages of the Device	41
Appendix C. Device Connectors	42

Package Contents

Device	1 pc.
This Manual	1 pc.
Stickers for MicroSD slot sealing	6 pc.

Introduction

Yacht Devices NMEA Bridge unifies two physical NMEA 2000 networks into a single logical network, smoothly exchanging messages between them. The Device also supports filtering and processing of transmitted messages.

This can accomplish the following tasks:

1. **Bypass the physical limits of NMEA 2000 networks** concerning length of networks (100 meters for regular cable and 250 meters for heavy or mid-type cable) and concerning the maximum number (50) of physical devices attached to the network. On a network with address capacity of 252, multiple bridges can be engaged to expand to around 250 physical devices.
2. **Isolate devices from each other.** Using the simple filter, you can block transmission of all or of selected messages from a given device in a separate subnet.
3. **Ensure proper functioning of equipment.** Correct the transducer offset of the depth sounder, or “delete” invalid data in messages from equipment that is only partially operational using a 2- or 3-line script.
4. **To ensure compatibility of equipment** from different generations. You can create and send any type of NMEA 2000 message using data from other messages in the network.
5. **Diagnose malfunctions in the NMEA 2000 network.** The Device can record network messages and debug data from custom programs on a MicroSD card in a text file. You can view the data in a standard text editor on a smartphone or tablet with a MicroSD slot, there is no need for a computer. You can even create and edit programs for the Device right on your phone!
6. **Safely connect devices** that do not meet NMEA 2000 standards. One of the CAN-interfaces on the device has high-voltage galvanic isolation and can operate at a higher supply voltage.
7. **Create gateways** for networks based on CAN protocol operating at a speed from 50 kbps to 1000 kbps. The programming language of the device is not designed for full-fledged applications, but one can create, for example, a gateway from a J1939 network to NMEA 2000.

Programming the device requires knowledge of NMEA 2000. Copy of NMEA 2000 standard can be purchased from the National Marine Electronics Association: <http://www.nmea.org>.



Yacht Devices would like to note that a NMEA 2000 network might contain important devices such as a deep sounder, magnetic compass and autopilot. Failure or incorrect operation of these devices can result in serious accidents and fatalities. When programming the Device, you must be fully aware of all the implications. Before making a sea-going trial, conduct mandatory training for the vessel's crew.

Warranty and Technical Support

1. The Device warranty is valid for two years from the date of purchase. If a Device was purchased in a retail store, when applying under a warranty case, the sale receipt may be requested.
2. The Device warranty is terminated in case of violating the instructions of this Manual, case integrity breach, repair or modification of the Device without manufacturer's written permission.
3. If a warranty request is accepted, the defective Device must be sent to the manufacturer.
4. The warranty liabilities include repair and replacement of the goods and do not include the cost of equipment installation and configuration, as well as shipping the defective Device to the manufacturer.
5. Responsibility of the manufacturer in case of any damage as a consequence of the Device operation or installation is limited to the Device cost.
6. The manufacturer is not responsible for any errors and inaccuracies in guides and instructions of other companies.
7. The Device requires no maintenance. The Device's case is non-dismountable.
8. If the event of a failure, please refer to Appendix A. before contacting the technical support.
9. The manufacturer accepts applications under the warranty and provides technical support only via e-mail or from authorized dealers.
10. Contact details of the manufacturer and a list of the authorized dealers are published on the website: <http://www.yachtd.com/>.

I. Product Specification

NMEA 2000 network connector, CAN 2
(SeaTalk NG, see note below)

NMEA 2000, CAN 1

Device's case

Bi-color LED

MicroSD slot

Cable, 500 mm

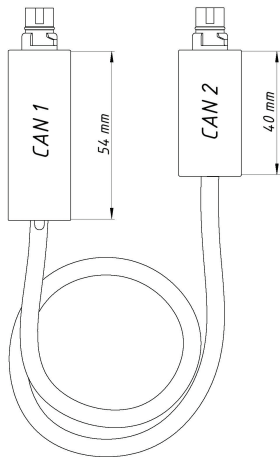


Figure 1. Drawing of YDNB-07R model of Bridge

Our devices are supplied with different types of NMEA 2000 connectors. Models with the suffix R at the end of model name are equipped with NMEA 2000 connectors compatible with Raymarine SeaTalk NG (as at the picture above). Models with the suffix N are equipped with NMEA 2000 Micro Male connectors (see Appendix C).

Device parameter	Value	Unit
Operating voltage, from CAN1 interface	9..16	V
Average current consumption, CAN1	38	mA
Load equivalency number, CAN1	1	LEN
Supply voltage of CAN2 interface	9..30	V
Average current consumption at CAN2	13	mA
Load equivalency number, CAN2	1	LEN
Isolation between CAN1 and CAN2	2500	V _{RMS}
Protection against reverse polarity	Yes	—
Operating temperature range	-20..55	°C
Cable length	500	mm
Device's case length (without connector)	54/40	mm
Weight without MicroSD card	52	g



Yacht Devices Ltd declares that this product is compliant with the essential requirements of EMC directive 2004/108/EC.



Dispose of this product in accordance with the WEEE Directive. Do not mix electronic disposal with domestic or industrial refuse.

II. Installation and Connection to NMEA 2000 Networks



Never connect both connectors of the Device to the same NMEA 2000 network. This can flood the network with infinite forwarding of messages and cause a temporary inoperability of the network.

The Device requires no maintenance. When deciding where to install the Device, choose a dry mounting location. Avoid places where the Device can be flooded with water; this can damage it.

The Device is directly connected to the network backbone without a drop cable. Before connecting the Device, turn off the bus power supply. Refer to the manufacturer's documentation if you have any questions regarding the use of connectors:

- SeaTalk NG Reference Manual (81300-1) for Raymarine networks
- Technical Reference for Garmin NMEA 2000 Products (190-00891-00) for Garmin networks

After connecting the Device, close the lock on the connection to ensure water resistance and reliability.

The microcontroller of the Device is powered by the CAN1 interface. The Device will not work until the NMEA 2000 network on the CAN1 interface is not powered up. If you want to start up the Device for familiarization purposes, the CAN2 interface can be left unconnected from the NMEA 2000 network.

The Device has a LED which flashes red or green. After turning the power in the NMEA 2000 network on, the Device's LED will give a series of 2 flashes 5 seconds apart. If this does not happen, see Appendix A.

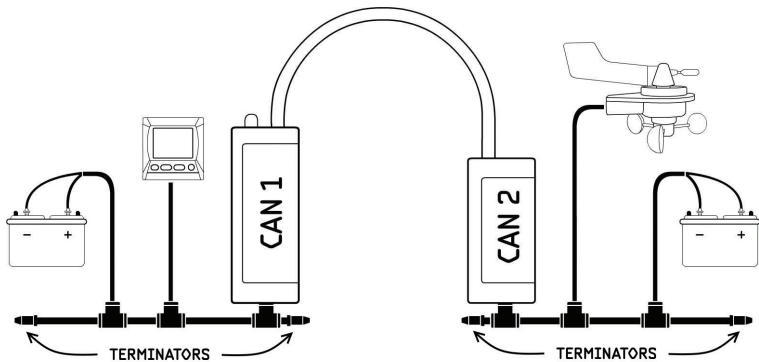


Figure 2. Scheme of a typical installation

Please remember that a NMEA 2000 network requires a separate power supply and a terminator on each side. If the Device is inset, dividing an existing NMEA 2000 network into two parts, you have to add a terminator to each of the segments and power the second segment with a 12V power supply. More information on this topic is available in the above listed documents from Raymarine and Garmin.

III. LED Signals

1. **Signal with period of 5 seconds, two flashes of the LED.** The first flash indicates the condition of the CAN1 interface network. The signal is green if within the last period (5 seconds) has been data received from the network or successfully sent, red if not. The second flash indicates the condition of the CAN2 interface network.

The Device can be configured to receive only a limited set of NMEA 2000 messages (see Section VII.3), the remaining messages are filtered at the hardware level. In this regard, some NMEA 2000 networks can indicate a red light much of the time, even when the network is functioning normally. In this case, to check the connection, turn one device that is on the network (e.g. the chart plotter) off and on again. The status of the network will be displayed with green flashes for some time as the device is powering up and connecting.

2. **Three flashes (colors may vary), one time after inserting the MicroSD card into the Device.** See Section V.
3. **Long flash (3-second), red or green.** Diagnostics mode started / finished, see Section X.
4. **Five green flashes when NMEA 2000 network is turned on.** The Device has the MicroSD inserted with a firmware update, the firmware is updated (see Section XI).

IV. MicroSD Slot and Card's Compatibility

The Device has a slot for a MicroSD card that allows you to configure the Device (see Section V) and update the firmware (see Section XI).



The Device slot has a 'push-push' mechanism that works on a spring and ensures proper card fixation. Improper loading or unloading (withdrawing your finger too quickly or not waiting for the click) can result in the card being propelled out of the Device up to 5 meters. To avoid possible eye injury, loss of or damage to the card, and other hazards, insert and remove the card with caution.

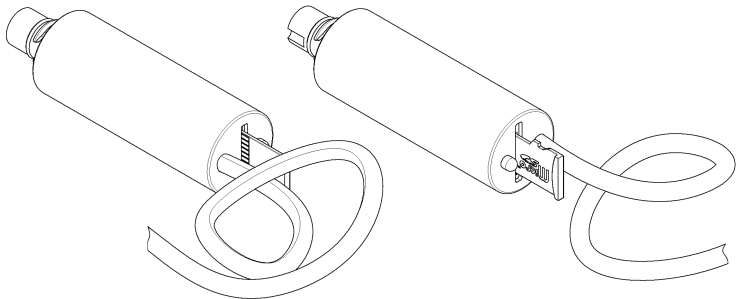


Figure 1. Device with MicroSD card (pin side visible at left, label side at right)

Since the MicroSD slot is usually not in use when the Device is working, we recommend sealing it with the sticker that is included with the Device or with a piece of tape to prevent water from entering the Device through the slot.

The Device supports MicroSD memory cards of all capacities and classes. The MicroSD card must be formatted on a personal computer before use in the Device. The Device supports the following file systems: FAT (FAT12, FAT16, MS-DOS) and FAT32. It does not support exFAT, NTFS, or any other file systems.

Be careful when inserting the MicroSD into the Device. The card is inserted with the label side toward the LED and with the pin side toward the cable.

V. Loading of Programs into the Device

Place YDNB.CFG file with the program to the root directory of a MicroSD card with a FAT or FAT32 file system. Insert the MicroSD card into the Device. After a few seconds, you should see the LED flash three times:

1. **Three red flashes** mean that the memory card cannot be read.
2. **A green followed by two red flashes** means that the YDNB.CFG file cannot be found on the memory card and the current Device configuration was saved to the YDNBSAVE.CFG file.
3. **A red followed by a green and another red flash** means, that the YDNB.CFG file contains errors and was not accepted by the Device. The text file YDNBERR.TXT was created in the root directory of the memory card, comprising an error log.
4. **Three green flashes** mean that the file has successfully been loaded into the Device. The text file YDNBSAVE.CFG was created in the root directory of the memory card, comprising the current program and the used settings.

The Device performs compilation of the program text into bytecode. Before the Device saves a program in the YDNBSAVE.CFG file, it decompiles the bytecode to text. This is why the contents of the YDNB.CFG and YDNBSAVE.CFG files can differ from each other.

The YDNB.CFG file must contain at least one interpretable line of code (a setting, filter etc.) without errors, to be loaded into the Device.

To modify the current program, insert a memory card into the Device that does not contain a YDNB.CFG file. The LED of the Device will flash green, red, red. This means that a YDNBSAVE.CFG file, comprising the current program, was saved onto the memory card. This program can be modified, saved as YDNB.CFG, and loaded back into the Device.

VI. Structure and Basic Syntax of the Program

The program defines algorithms and rules for the processing and forwarding of NMEA 2000 messages that the Device receives via the CAN1 and CAN2 interfaces.

The program consists of settings, subprograms of filters, and comments. The settings and filters are described in detail in the later sections of this Manual.

Comments in the program are added after the # symbol. Comments can be situated at the beginning of a line as well as after interpretable program text.

Settings can be set anywhere in the program, except for inside subprograms of filters. Nevertheless, we recommend declaring settings before filters.

Example 1.

```
# Example N1
FW_CAN1_TO_CAN2=ON           # Allow forwarding of all mismatched messages
FW_CAN2_TO_CAN1=OFF         # Setting for other direction
match(CAN2,0x01FD0600,0x01FFFF00) # 1st filter
{
    # Empty subprogram, matched messages will be dropped
}
match(CAN2,0x00000010,0x000000FF) # 2st filter
{
    send(CAN1)                # Forward of matched message to CAN1 interface
}
match(CAN1,0x00000020,0x000000FF) # 3nd filter (for CAN1)
{
    # No send(), matched messages will be dropped
}
# End of program
```

A filter consists of a header, which begins with the keyword `match()`, and contains the data for the matching of a message, as well as a processing subprogram entered in the specially created programming language.

The order of the filters is important because the Device matches received messages with the filters, in the order they are specified in the program. In case of a match, the message is sent to the processing subprogram of the filter, which will then be responsible for forwarding or blocking the message. The message will not be compared with the next filter.

In case there is no match with any of the filters, the Device follows the settings. If message forwarding is enabled for the interface (by default, yes), the given message will be sent to the other CAN interface. If disabled, the message will be discarded.

The program in example 1 comprises two settings and three filters. The Device will forward every message it gets from the CAN1 interface to the CAN2 interface, except for messages sent by the device with address 0x20. But only messages with the address 0x10 will be forwarded from the CAN2 to the CAN1 interface, with the exception of messages with PGN 0x1FD06.

Looking ahead, we explain the operation of this program. When receiving a message from the CAN2 interface, it coincides with the first filter of the program when its PGN is equal to 0x1FD06. The processing subprogram of the first filter is empty, so the message will not be forwarded and the processing of the message is completed.

In case there is no match with the first filter, comparison with the second filter will be executed. It is successful if a device with the address 0x10 at the CAN2 network sent the message. The processing subprogram of this filter consists of only a call of the `send()` function, which initiates the forwarding of the message.

If the messages received from the CAN2 interface do not coincide with the first two filters (the third filter only affects messages received from the CAN1 interface), the messages will be blocked according to the `FW_CAN2_TO_CAN1` setting.

VII. Description of the Settings

Note that a vertical bar (pipe) is used in the descriptions below to separate alternative setting values. ON|OFF means, that the setting can have to different values — ON or OFF.

1. Forwarding of messages

```
FW_CAN1_TO_CAN2=ON|OFF  
FW_CAN2_TO_CAN1=ON|OFF
```

Enables or disables the automatic forwarding of messages that do not coincide with filters between the CAN1 and CAN2 interfaces, see example 1 at page 15.

2. Assembly of NMEA 2000 fast messages

```
PGNS_TO_ASSEMBLY=x
```

x — from one to five PGN, entered as decimal or hexadecimal values, separated by commas.

NMEA 2000 messages with a length from 9 to 223 byte, are transmitted in a series of standard CAN messages with a length from 1 to 8 byte (see „3.1 Fast-packet messages“ in the NMEA 2000 standard documentation).

The Device can pre-assemble NMEA 2000 fast messages from series of CAN messages and transmit them to the program in an assembled state. In this parameter, you can select up to five messages types for assembly, using PGN. Messages not completely assembled will be discarded.

When such a message is send, it is dissembled into a series of CAN messages and put into a sending queue. It is recommended not to use the assembly of NMEA 2000 messages when possible, but to process them at the level of CAN messages, as this will significantly reduce the sending delay in the Device (see Section IX).

3. Hardware filters

```
CAN1_HARDWARE_FILTER_y=f, m  
CAN2_HARDWARE_FILTER_y=f, m
```

y – number of the hardware filter, decimal number from 1 to 7;
f – filter value, decimal or hexadecimal number (29 significant bits);
m – filter mask, decimal or hexadecimal number (29 significant bits).

The Device can filter messages from the CAN1 and CAN2 interface at hardware level, which in some cases can reduce the load on the microprocessor by a factor of up to a 100. The message selection is carried out through the standard 29-bit identifier of CAN messages, which contain the message priority, PGN, the sender's address and (in some cases) the recipient's address.

Messages are only passed to the program if they match one of the hardware filters. It is possible to set up to 7 custom hardware filters for each interface, with numbers from 1 to 7.

Additionally, the Device has a set system hardware filter for each interface with the number 0, which cannot be modified by the user:

```
CAN1_HARDWARE_FILTER_0=0x00E80000, 0x01F90000  
CAN2_HARDWARE_FILTER_0=0x00E80000, 0x01F90000
```

The filter (first parameter) sets bits for the comparison with the message identifier and the mask (second parameter) indicates the bits whose comparison result is significant.

Thus, the system hardware filter passes only messages with the following PGNs: 0xE800 (ISO Acknowledgement), 0xEA00 (ISO Request), 0xEC00 (ISO Transport protocol), 0xEE00 (ISO Address Claim).

If no custom hardware filter for the interface is defined in the program, a filter that accept all messages will be automatically added for the given interface:

```
CAN1_HARDWARE_FILTER_1=0, 0
```

So if no custom hardware filter is set in the program, all messages will be passed to the program.

4. NMEA 2000 instance

```
DEVICE_INSTANCE=x  
SYSTEM_INSTANCE=y
```

x – number from 0 to 255, *y* - number from 0 to 15.

These settings allow you to program the device and system instances of the Device, used in the Device information (NAME) and transmitted in the ISO Address Claim message.

The default value of both settings is 0. These settings will not be saved to the YDNBSAVE.CFG file if they have the default values.

5. Message slots initialization

```
SLOTx=aabbccdd..  
      or  
SLOTx=aa bb cc dd...
```

x – number of slot, 1 – 3;
aabbccdd – sequence of bytes (1 – 229), hexadecimal values.

These settings are used to initialize slots when the Device is powered up (or when a new program is loaded from the MicroSD card). Data from slots may be loaded to the current message buffer using the `load()` function (see VIII.8). The buffer's format is described in VIII.2. You can overwrite slot content in the program filters.

Example of non-addressed ISO Request of ISO Address Claim:

```
SLOT1= 00FFEA18 FF 03 00EE00
```

6. Speed of the CAN2 interface

CAN2_SPEED=x

x - the speed in kbps, factory setting is 250, allowed values are 50, 125, 250, 500 and 1000.

The speed of the CAN2 interface can be set within a range from 50 to 1000 kbps (NMEA 2000 has 250 kbps speed). This allows interfacing with various CAN networks. The speed of the CAN1 interface cannot be changed, it has fixed speed of 250 kbps.

VIII. Software Filters

To write processing programs for messages, we created a special programming language similar to many other modern languages, but with a number of limitations imposed by tasks and computing capabilities of the microcontroller.

As you may have noticed in Example 1, the interpreted lines of the program contain no special symbols that indicate the end of a line (in C++, Java, and JavaScript such symbols are semi-colons). Each new interpreted expression must start on a new line.

All numbers used in the program are integer values, specified as decimal or hexadecimal numbers. The case of hexadecimal numbers is not important, but the 0x prefix must be indicated in lower case.

Numeric values may also be shown in 7-bit ASCII, a symbol must be enclosed in single quotes: 'a', '1', etc.

Preset constants used in the program (DATA), interface identifiers (CAN1, CAN2), data type identifiers (UINT32, FLOAT, etc.), all are case sensitive and must be indicated in upper case.

The programming language does not support creation of user functions, but does allow use of built-in functions (`get()`, `set()`, `cast()`, `send()` and others).

1. Defining filters

A filter is defined by the keyword `match()` with 3 parameters in parentheses, after which its subprogram follows in curly brackets (which may be empty):

```
match(interface_id, filter, mask)
{
}
```

interface_id — the interface identifier: CAN1, CAN2 or ANY;
filter and *mask* — the numeric values of the filter and mask.

The Device compares the filter parameters with the interface from which the message is received, and compares the 29-bit identifier of the CAN message (which contains the priority of the message, the PGN,

sender address, and [in some messages] the address of the recipient) with the filter and mask. The filter (the second parameter) assigns bits for comparison with the message identifier, and a mask (the third parameter) indicates bits whose comparison result is significant.

So, the filter:

```
match(CAN1,0x00E80000, 0x01F90000)
```

matches only messages received from the CAN1 interface and which have the following PGN: 0xE800 (ISO Acknowledgement), 0xEA00 (ISO Request), 0xEC00 (ISO Transport protocol), 0xEE00 (ISO Address Claim).

The keyword `match()` is not translated into bytecode, so the comparison of the specified filters with the messages received is very quick.

The program may contain up to 20 filters.

2. Message buffer

The subprogram has access to the message and its 29-bit CAN identifier that is stored in the buffer described in Table 1.

Table 1. Message buffer structure

Offset	Length in bytes	Description
0	4	29-bit identifier CAN message (LSB)
4	1	0xff – indicator of a single-frame NMEA 2000 message (CAN message with 29-bit identifier); 0xfe - indicator of a CAN message with an 11-bit identifier; 0x00..0xE0 in steps of 0x20 – indicator of the NMEA 2000 fast message (sequence counter); other values are reserved
5	1	Message length (3..223)
6	223	Message data

By modifying the values in the buffer, the subprogram can modify messages and even create new messages. Regardless of the actual length of the received message, the subprogram has access to all 229 bytes of the buffer.

Please note that if a NMEA 2000 fast message's PGN is not included in the PGNS_TO_ASSEMBLY list, the program will be called for each CAN message in the sequence, and the byte at offset 4 will be set to 0xff.

3. Sending a message

To send message stored in the buffer a built-in `send()` function is used:

```
send(interface_id)
```

interface_id — interface through which the message is sent, CAN1 or CAN2. This parameter can be omitted, in this case the message will be sent through the interface opposed to the one from which the processed message was received.

In order to avoid embarrassing mistakes with unpredictable consequences, we recommend always to use the `send()` without a parameter, wherever possible.

Note that the filter subprogram is responsible for forwarding the message being processed. If your subprogram does not contain a call to `send()`, then the processed message will not be forwarded.

4. Variables, operators, and expressions

There are 26 global variables available to the programmer, with names from A to Z. The case of the name of the variables is not important. Variables can be one of the following types that are defined implicitly and automatically converted to the necessary type when operations are preformed.

- INT8 — signed 1-byte integer;
- UINT8 — unsigned 1-byte integer;
- INT16 — signed 2-byte integer;
- UINT16 — unsigned 2-byte integer;
- INT32 — signed 4-byte integer;

- `UINT32` – unsigned 4-byte integer;
- `FLOAT` – 4-byte number with floating point.

When you turn on the Device, all variables are initialized to 0 type `INT32`. You can use this fact for the initialization of variables.

The following arithmetic, logical and shift operations are allowed (listed in order of priority):

- `*` (multiplication), `/` (division), `%` (the remainder of integer division)
- `+` (addition), `-` (subtraction), `<<` (binary shift to the left), `>>` (binary shift to the right), `|` (logical OR), `^` (XOR), `&` (logical AND)
- `=` (assignment)

Parentheses are used to increase the priority of the operation:

```
A = (3 + 5) * 2 # Equivalent A = 16
```

When you assign values to variables, the result type is automatically selected:

```
A = 1 # INT32
B = 2.0 # FLOAT
C = '1' # 0x31, UINT8
D = B * C # 0x31 * 2.0 = 49 * 2.0 = 98.0, FLOAT
E = 0xFF00AA3F # UINT32
```

Use the built-in `cast()` function to set an expression to a specified type:

```
[type] cast(expression,type)
```

Where *type* is one of the types listed above. For example:

```
B = cast(1.5, INT8) # 1, INT8
C = cast(B << 2, FLOAT) # 4.0, FLOAT
```

Note that the compiler does not optimize algorithms, try to perform calculations efficiently.

5. Modification of the message buffer

To modify a message in the buffer and read data from it, there are, respectively, two built-in functions `set()` and `get()`:

```
set(expression1,type,expression2)
[type] get(expression1, type)
```

expression1 — expression converted to `UINT8` type, offset of the first byte of data in the buffer;

type — the data type identifier (see VIII.4);

expression2 — an expression, the result of which is will be casted to the specified type and placed in a buffer with the offset that is specified in *expression1*.

Addressing of the data in the buffer starts at zero. For more convenient access to the data, we recommend using the built-in constant `DATA`, which is equal to the offset in the buffer of the first byte of the message's data.

Example 2.

```
match(CAN1,0x1F50B00,0x1FFFF00)
{
    A = get(DATA+1, UINT32)    # get 4-byte integer from message
    set(DATA+1, UINT32,A + 20) # set the corrected value
    send()                    # send message to CAN2
}
```

In Example 2, a filter for Water Depth messages (PGN 0x1F50B) is implemented. The subprogram retrieves the value of the depth (four-byte integer value, bytes 1-4 of the message's data), and returns it, adding 20. After that, the message is sent. The result of the program is to increase the sonar readings by 20 centimeters.

6. Comparison operator `if()`

The comparison operator *if* is the only operator in the language that supports comparisons: `>` (more than), `<` (less than), `==` (equal), `!=` (not equal), `<=` (less than or equal), `>=` (more than or equal). The *else* block is optional:

```

if (expression1 operator expression2) {
}
else {
}

```

The above Example 2 does not take into account that the transmitted sonar depth can be set to 0xFFFFFFFF, which means "no data." Using the operator `if()` we can check the data for the admissibility of a range of values:

Example 3.

```

match(CAN1,0x1F50B00,0x1ffff00)
{
    a = get(DATA+1, UINT32)           # get 4-byte integer from message
    if (A < 0xFFFFFFFF-20)           # is the value valid?
    {
        set(DATA+1, UINT32,A + 20) # set the corrected value
    }
    send()                             # send message to CAN2
}

```

7. Real time functions

The majority of data in a NMEA 2000 network has a limited life span. If you want to store data in global variables to be used in filters of other messages, you need to determine their relevance to have the filters work correctly.

```

UINT32 timer()
UINT32 timediff(expression)

```

The `timer()` function returns the number of milliseconds since the Device was turned on. The value of the timer overflows and is reset approximately every 49 days. To calculate the difference (in milliseconds) between the current timer value and the stored value of the timer, it is convenient to use the built-in `timediff()` function, which handles timer overflows and guarantees the return of the correct value if the difference between the current timer value and the value transferred is less than 49 days. See also: `heartbeat()`, VIII.11.

8. Storage buffers

```
save(slot)
load(slot)
```

The `save(slot)` function stores data from the message buffer in one of the three additional buffers, and the function `load(slot)` overwrites the current message buffer with the data from specified slot. The slot identifier can have one of three values: `SLOT1`, `SLOT2`, `SLOT3`. You may set slot contents in settings (see VII.5).

9. Getting the Bridge address

```
UINT8 addr()
```

This function returns the address of a NMEA 2000 device. The address of the device is the same on both the `CAN1` and `CAN2` interfaces.

In some cases, the normal operation of the network can be broken when a message with the address of a different device is sent. In this case, you can send the message using the Bridge address. This contradicts NMEA 2000 standards, as the PGN of your message is not declared in the list of messages transferred by the Bridge, and in rare cases, the message may be ignored by some devices, but it does not affect operation of other devices.

Example 4.

```
# Processing of Actual Pressure messages (PGN 0x1FD0A) and generation of the
# Environmental Parameters messages (PGN 0x1FD06)

match(CAN2,0x1FD0A00,0x1ffff00)
{
    send()                # Forward the original message
    A = get(DATA + 2, UINT8) # Extract the data type
    if (A == 0)           # Is it atmospheric pressure?
    {
        B = get(DATA + 3, INT32) # Extract the pressure value
        if (B != 0x7FFFFFFF) # Check the validity of the value
        {
            set(1, UINT8,6) # Change PGN from 0x1FD0A to 0x1FD06
            set(DATA + 1, UINT32, 0xFFFFFFFF) # Set the unused fields
            set(DATA + 5, UINT16, B / 1000) # Convert and set pressure
            send(CAN1) # Send Environmental Parameters to CAN1
            set(0, UINT8, addr()) # Replace the address with the address
            # of the Bridge
            send(CAN2) # Send Environmental Parameters to CAN2
        }
    }
}
```

In the above example, the program sends out a message from CAN2 to CAN1 for Actual Pressure (PGN 0x1FD0A). For compatibility with older hardware, the program also prepares an Environmental Parameters message (PGN 0x1FD06) and sends it to CAN1 with the original address, and to CAN2 (where the original sender is located) it sends the same message, but using the address of the Bridge as the sender in that subnet.

10. Program initialization

Like filters, the initialization section is defined by the keyword `init()` without parameters, after which its subprogram follows in curly brackets:

```
init()
{
}
```

`init()` can be used to initialize program variables; it is guaranteed to be called when the Device is turned on, before any other user code. Note, the program inside `init()` can't send network messages because the NMEA 2000 standard prohibits sending of any messages during the first 250 milliseconds after the device's address claim procedure, and any `send()` calls from user code will be ignored.

11. Heartbeat

Like filters or `init()`, `heartbeat` is defined by the keyword `heartbeat(ms)` with one numeric decimal parameter which specifies the periodic interval in milliseconds, after which its subprogram follows in curly brackets:

```
heartbeat(1000)
{ # This code will be executed every second
}
```

`heartbeat()` can be used when a code must be executed periodically. The `heartbeat()` is first called from the user's program immediately after `init()`.

IX. Optimization and Performance

The transfer time of single-frame NMEA 2000 messages is about 520 microseconds. The Device does not waste computing resources to receive and transmit messages; these operations are performed by the hardware.

The time from receipt of a message to the time that it is sent, including the execution of the program consisting of 20 filters, for messages that do not match any of the filters is 100 microseconds. Thus, message transfer between the two interfaces is possible in real time with a minimal delay.

Of the 100 total microseconds, only 40 microseconds are used for the comparison with filters (program execution). The rest of the time is spent processing interruptions from the CAN controller, placing the received message into the input queue, removing it from the queue for the program, and then handing it over to the CAN controller to send.

The runtime (for messages matching a filter) of Example 2 is 240 microseconds, and of Example 7 – 515 microseconds. The total delays in the Device are 300 and 575 microseconds respectively.

NMEA 2000 messages have a period of 100 milliseconds or more, loading of the NMEA 2000 network is usually 10-30%. So it is difficult to imagine conditions under which a delay of 0.5 milliseconds for one message may cause a significant queue.

None the less, the Device does have a software queue for 100 inbound and 100 outbound messages, with a maximum lifetime of messages in the queue of 50 milliseconds.

The device's performance is sufficient for practical applications, even in highly loaded networks. As long as egregious programming errors are avoided, no overload issues should occur.

1. Using hardware filters

The most common scenario for using the Device is where there are one or two devices on one interface (e.g. CAN2) whose messages need to be processed through some filters. At the same time, all the other devices are on a different interface (CAN1). In a such case, the program shown in Example 5 below is extremely effective.

Example 5.

```
FW_CAN1_TO_CAN2=ON
FW_CAN2_TO_CAN1=ON
CAN1_HARDWARE_FILTER_1=0x00000000, 0x00FFFFFF

match(CAN2,0x1FD0A00,0x1ffff00)
{
    # some required processing here...
}
```

Thanks to the system hardware filters (see VII.3), messages which are required for the normal function of the NMEA 2000 network (ISO Acknowledgement, ISO Request, ISO Address Claim) will be transmitted through the hardware filter to both interfaces.

Because there is no custom hardware filter for the CAN2 interface in the program, the Device will add a hardware filter that allows all messages from this interface (see VII.3) to be passed to the program. The filter CAN1_HARDWARE_FILTER_1 will not allow any additional messages; we added it only to ensure that the Device would not add a filter to the CAN1 interface that allows all messages through.

So the Device will not waste computational resources on processing and send “unneeded” messages from the CAN1 network to the CAN2 network. In a typical network such messages can comprise over 99% of the total.

2. match() instead of if()

There is the temptation to use the `if()` operator to process messages with different PGNs inside a single filter.

The comparison of the 29-bit identifier of received message and the filters is not performed from bytecode, and is executed several times faster than would be possible in the subprogram. Increasing the number of filters has practically no effect on the time needed to compare identifier with the filters (see Example 6at the next page).

Incorrect Example 6.

```
match(CAN1,0x00000010,0x000000FF) # sent from 0x10 address?
{
    p = (get(0,UINT32) >> 8) & 0x1ffff # get the PGN
    if (p == 0x1FD0A) # is it "Actual Pressure" message?
    {
    }
}
```

3. Optimization instead of a compiler

In this version of the compiler there are no optimization algorithms, and the bytecode for the following expressions will contain multiplication and division operations:

$$A = (6/3) * 5$$

The compiler will create code to calculate $B*C+D/5$ two times for the following program:

```
A = B*C+D/5
F = B*C+D/5 + 1
```

If you have a lot of calculations in your program, try to optimize them.

Future updates of the software will place special focus on issues of optimization during compilation to bytecode.

4. Do not use fast messages assembly

In most cases, the data in the message can be modified without pre-assembly. Consider, for example, hacking the log distance (a criminal offense in most countries) using the Distance Log message (PGN 0x1F513).

Example 7.

```
PGNS_TO_ASSEMBLY=0x1F513
match(CAN1,0x1F51300,0x1FFFF00)
{
    set(DATA+6, UINT32, get(DATA+6,UINT32) - 1852000) # 1000 nm of
    send()
}
```

Example 8.

```
match(CAN1, 0x1F51300,0x1FFFF00)
{
    if (get(DATA,UINT8) & 0x1F == 1) # 2nd message is sequence?
    {
        set(DATA+1, UINT32, get(DATA+1,UINT32) - 1852000) # 1000 nm off
    }
    send()
}
```

The result of the code is the same in both examples. In the second example, the subprogram will be performed three times, as the NMEA message with PGN 0x1F513 is sent in three CAN messages. The code execution time in the second example is longer, as it has additional operations.

But in the case of Example 7, the Device will receive all three CAN messages first and only then transmit the assembled NMEA message to the program. The time between receiving the first and third messages will be over 1000 microseconds, the period in which two messages on the CAN network must be transmitted. When you call the send(), the NMEA 2000 message will be divided into three CAN messages and go to the send queue. The third message will, at the earliest, be sent in 1000 microseconds.

In this way, in Example 7, all messages will be sent approximately 1000 microseconds earlier than in Example 8. In practice, the difference of 1 millisecond may not seem significant, but the longer the NMEA 2000 message, the longer will be the gap. Furthermore, if there is high load on the either of the CAN networks, the program shown in Example 7 has an additional advantage.

X. Debugging of the Program



The debug mode is not a normal operation mode of the Device. The recording of debugging data can lead to additional message delay and the loss of NMEA 2000 messages.

To activate the debug mode, add the following setting to the program file:

```
DIAGNOSTICS=x
```

Where x is the duration in seconds for which the debug mode is active.

Note that this setting is not saved to the Device volatile memory and is only active until either the MicroSD card is removed or the Device is turned off.

After three LED signals that indicate the processing of the YDNB.CFG file, the LED will glow green for 3 seconds. This signal means that the debug mode is activated.

The log file YDNBLOG.TXT will be created (overwritten) on the memory card. It will contain all NMEA messages matched with filters; and all messages which are sent by the `send()` function from filter subprograms. Besides that, you can use the built-in this function:

```
log(expression)
```

which save the expression result or value of variable into the log file.

Do not remove the memory card from the Device while the debug mode is active!

After the specified duration, the Device's LED will glow red for 3 seconds. This means that the debug mode is deactivated and the log file is closed. Now you can safely remove the memory card from the Device.

Example of the YDNBLOG.TXT file content:

```
00:30.310 RX CAN1 FILTER 01, 0x09FD0205 1A1A018544FAFFFF
00:30.310 LG CAN1 FILTER 01, INT32 19400
00:30.310 TX CAN2 FILTER 01, 0x09FD0205 1A1A018544FAFFFF
```

RX and TX are the received and sent messages with an indication of an interface and the number of the matched filter. LG is the record created by the `log()` function, with the type and value of an expression or variable.

You can record every message received by the Device to a log file with the following filter:

```
match(ANY, 0, 0)
{
    send()
}
```

Do not forget to comment out all the `log()` function calls after you have finished debugging. Independent of the `DIAGNOSTICS` setting in the program, the `log()` functions will be compiled into the bytecode and will consume computing resources.

The Bridge supports binary log files also. You may turn on use of binary logs in the program text (this setting is also not saved to the Device's volatile memory):

```
LOG_FORMAT=BINARY
```

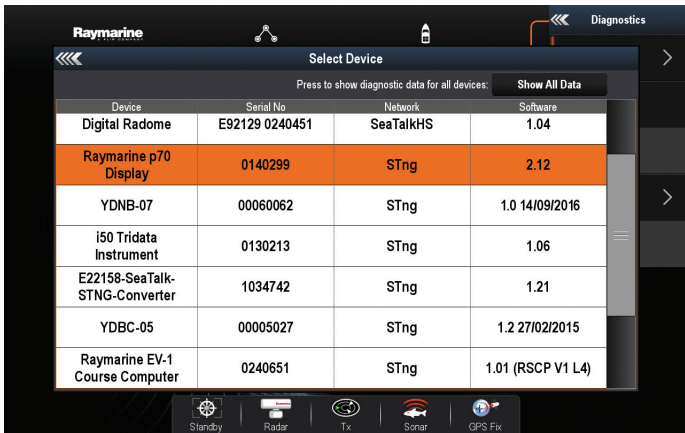
The `TEXT` value forces text log format, which is used by default. Binary logs are stored in files with a `.CAN` extension. Unlike text logs, which contain only messages matched with filters (or sent from filters), binary logs contains all messages received or transmitted on both CAN interfaces. And unlike text logs, you can't use the `log()` function to save a variable's value to the binary log file.

Binary logs may be opened, viewed, converted or exported to other formats with the free CAN Log Viewer, which works on Microsoft Windows, Linux and Max OS X. You may download it from our web site. The `.CAN` format is open and described in the CAN Log Viewer documentation.

XI. Firmware Updates

In the root folder of the MicroSD card with FAT or FAT32 file system, copy BUPDATE.BIN, which contains the firmware update of the Device. Insert the card into the Device and turn on the power in the NMEA 2000 network.

From 5-15 seconds after powering on, the LED will flash 5 times with green light. This indicates that the firmware update is successfully completed.



Device	Serial No	Network	Software
Digital Radome	E92129 0240451	SeaTalkHS	1.04
Raymarine p70 Display	0140299	STng	2.12
YDNB-07	00060062	STng	1.0 14/09/2016
i50 Tridata Instrument	0130213	STng	1.06
E22158-SeaTalk-STNG-Converter	1034742	STng	1.21
YDBC-05	00005027	STng	1.2 27/02/2015
Raymarine EV-1 Course Computer	0240651	STng	1.01 (RSCP V1 L4)

Figure 4. Raymarine c125 MFD devices list with Bridge (YDNB-07)

If the Device already is using the given version of the firmware, or if the Device cannot open the file or the file is corrupted, the boot loader immediately transfers control to the main program. This is done without visual cues.

The Device information including the firmware version is displayed in the list of NMEA 2000 devices (SeaTalk NG, SimNet, Furuno CAN) or in the common list of external devices on the chart plotter (see third line at Figure 4). Usually, access to this list is in the "Diagnostics", "External Interfaces" or "External Devices" menu of the chart plotter.

Index

- addr(), function26,27
- ANY, identifier20,34
- buffer, see "message buffer".....21,24,26
- BUPDATE.BIN, file35
- CAN, file format34
- CAN1, identifier15,20,22
- CAN1_HARDWARE_FILTER_x, setting18,30
- CAN2, identifier15,20,22
- CAN2_HARDWARE_FILTER_x, setting18
- CAN2_SPEED,setting.....20
- cast(), function23
- custom hardware filter18,29
- DATA, constant20,24
- DEVICE_INSTANCE, setting19
- DIAGNOSTICS, debug setting33
- else, operator24,25
- filter, see "software filter"15,20
- FLOAT, type20,23
- FW_CAN1_TO_CAN2, setting15,17,30
- FW_CAN2_TO_CAN1, setting15,17,30
- get(), function24
- hardware filter18,29
- heartbeat().....28
- if(), operator24, 30
- init()28
- INT8..INT32, type22,23
- load(), function26
- log(), function33,34
- LOG_FORMAT, debug setting34
- match(), filter15,20
- message buffer21,24,26
- operators23,24
- PGNS_TO_ASSEMBLY, setting ... 17,22,32
- save(), function26
- send(), function15,22
- set(), function24
- software filter15,20
- system hardware filter18,30
- SLOTx, setting19
- SYSTEM_INSTANCE, setting19
- timediff(), function25
- timer(), function25
- typeofvariable22,23
- UINT8..UINT32,type.....22,23
- variables22,23
- YDNB.CFG,file.....14,33
- YDNBERR.TXT, file14
- YDNBLOG.TXT, file33,34
- YDNBSAVE.CFG, file14,19

Appendix A. Troubleshooting

Situation	Possible cause and correction
The LED does not signal after the NMEA 2000 network is powered on	<ol style="list-style-type: none"><li data-bbox="318 149 1113 239">1. Make sure that you turned on the NMEA 2000 network to which the CAN1 interface of the Device is connected. The microcontroller of the Device is powered by the network the CAN1 interface is connected to.<li data-bbox="318 242 1113 332">2. No power supply on the bus. Check if the bus power is supplied (NMEA 2000 network requires a separate power connection and cannot be powered by a plotter or another device connected to the network).<li data-bbox="318 335 1113 439">3. Loose connection in the power supply circuit. Treat the Device connector with a spray for cleaning electrical contacts. Plug the Device into another connector.
The state of CAN1 or CAN2 network always is "red"	<ol style="list-style-type: none"><li data-bbox="318 442 1113 532">1. No power supply on the bus. Check if the bus power is supplied (NMEA 2000 network requires a separate power connection and cannot be powered by a plotter or another device connected to the network).<li data-bbox="318 535 1113 605">2. Loose connection in the data circuit. Treat the Device connector with a spray for cleaning electrical contacts. Plug the Device into another connector.<li data-bbox="318 608 1113 698">3. There are problems in the NMEA 2000 network. Make sure that both terminators are installed in each network (see Section II). Plug another device into the selected connector and make sure it functions.<li data-bbox="318 701 1113 781">4. Most messages are discarded by hardware filters. Turn any device in the network off and on again (see III.1).
A loaded program is not working properly.	Debug the program. See section X.

Situation	Possible cause and correction
The Device LED flashes “green” every five seconds, but the Device is not displayed in the list of external devices on the plotter.	There are problems in the NMEA 2000 network. The network segment is not connected to the plotter or there are missing terminators in the network or the network is too long. Plug another device into the selected connector and make sure it appears in the list of devices on the plotter.

Appendix B. List of NMEA 2000 Messages of the Device

Messages from the table below are received independently from any settings, hardware filters (see VII.3) or the program. All received messages, including the messages that are addressed to the Device, are transmitted to the program and may be forwarded to another CAN interface by it. The Device processes messages that are addressed to it after they have been processed by the program (but independently of the program's results). The Device's answers to such messages are not passed to the program, but are sent directly to both CAN interfaces.

Table 1. Supported NMEA 2000 messages

PGN	Transmit	Receive	Description
59392	Yes	Yes	ISO Acknowledgment
59904	—	Yes	ISO Request
60928	Yes	Yes	ISO Address Claim
126464	Yes	—	PGNs Group List
126996	Yes	—	Product Information

Appendix C. Device Connectors

V+, V- - Battery 12V; CAN H, CAN L - NMEA 2000 data;
SCREEN - Not connected in the Device.

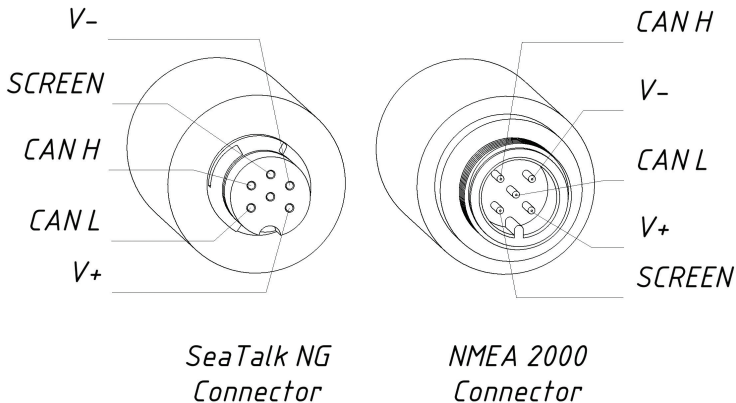


Figure 1. NMEA 2000 connectors of the YDNB-07R (left) and YDNB-07N (right) models

