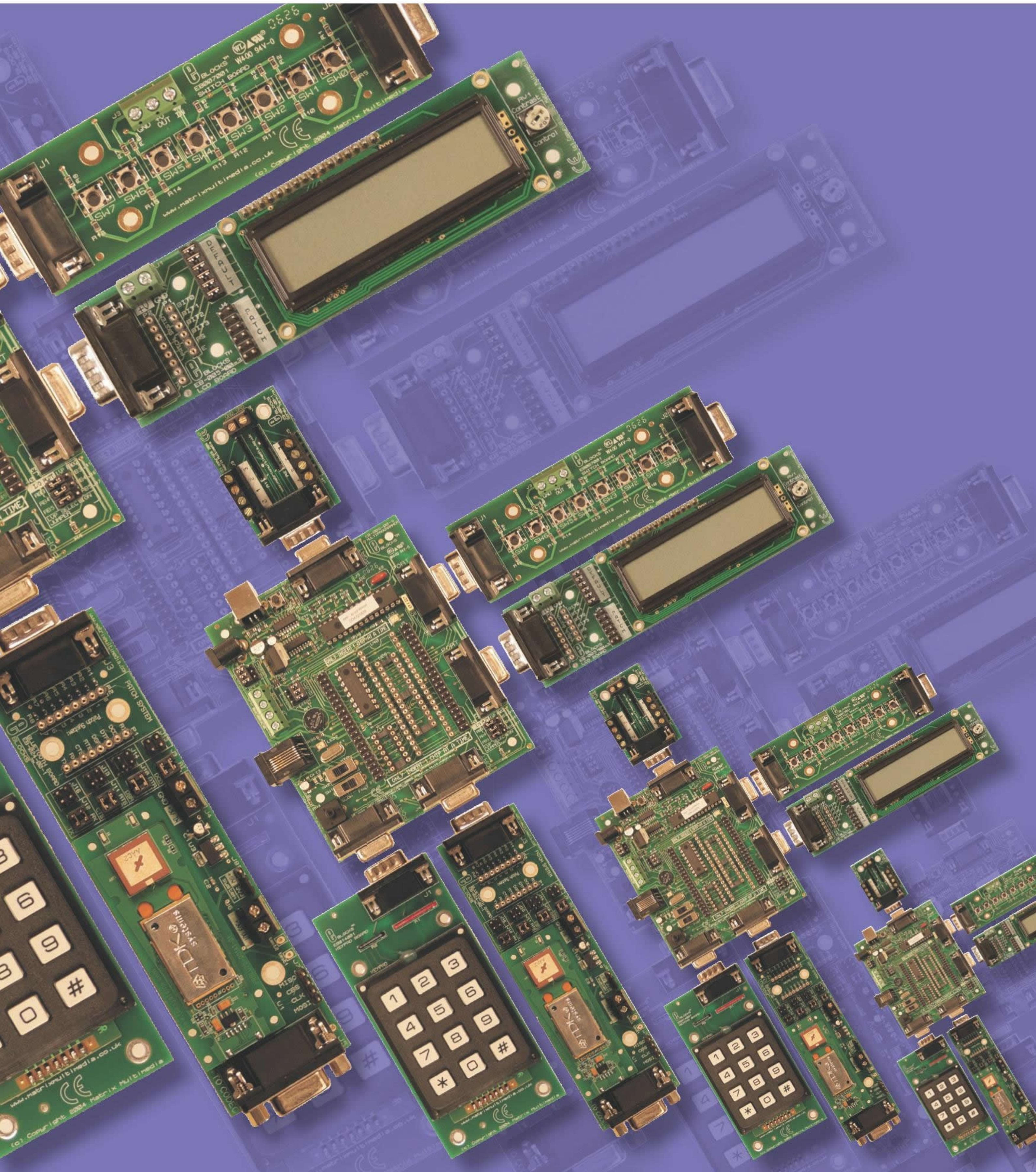
MATRIX

getting started guide

E-Blocks Easy CAN Bus Bundle



Installing Flowcode

Instruction for installing Flowcode can be found inside the installation booklet located inside the Flowcode DVD case.

Before starting with the course it is recommended to update your version of Flowcode to the latest released version. This allows for the latest bug fixes and components to run on your machine. The latest version of Flowcode can be found by visiting the Matrix TSL website and clicking on the Flowcode page.

http://www.matrixtsl.com/Flowcode3a-X.php

Getting Started with Flowcode

There is a free online course available for helping with getting started with learning Flowcode. This course covers basic principals through to designing your own programs and programming the devices. It is recommended that you take time to go through this course before proceeding with the bundle exercises to give you a better grasp of what the Flowcode program is doing.

The online course is available from the learning centre on our website or by visiting the following address:

http://www.matrixtsl.com/lc_microcontroller.php

Flowcode Examples

A number of pre-made example files are available for download from the main Flowcode page on the Matrix TSL website. These files are also located on the Flowcode CD. Before the example files can be used you must first copy them into a folder on your hard drive.

The example programs referenced in this bundle can be found at the following web address:
http://www.matrixtsl.com/lc_bundle_manuals.php

Or by clicking the bundle manuals link from the Learning Centre area of our website.

Flowcode Help

There is a help file available that covers all the main features of Flowcode. This help file can be accessed by clicking the question mark icon in the main Flowcode toolbar or alternatively clicking the help menu and selecting contents.

There are also help files available for each and every component in Flowcode which explain the functionality of the component and the component macros. The component help files can be found by selecting the component on the panel and the clicking the Help button in the properties toolbar.

General Support

Support for frequently encountered problems can be found online on our FAQ’s site.
Our online forums can also be used as a general discussion area or for help or advice.

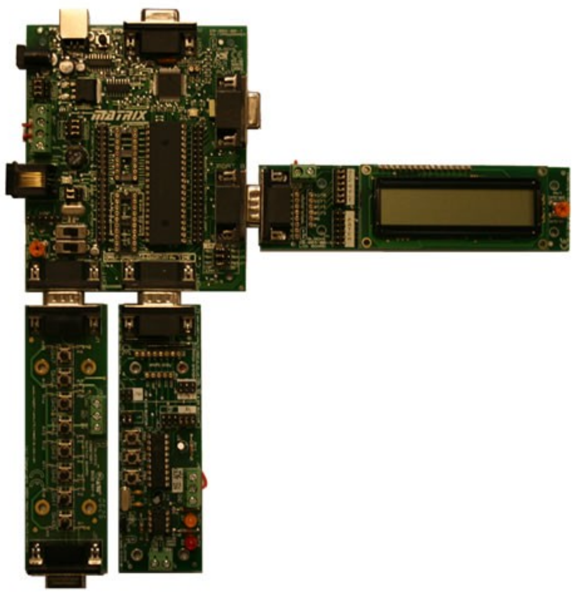
Wiring & Testing

EB984-80-1

E-Blocks Easy CAN BUS Bundle

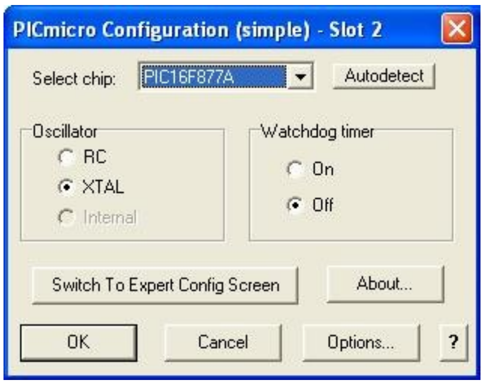
To set up your E-Blocks for use with the example programs you must perform the following actions:

- Remove the 16F88 devices from the EB006's;
- Insert the 16F877A's into the EB006's;
- On each Multiprogrammer the CAN E-Block must be connected to Port C;
- The LCD E-Block should be connected to Port B of node 1;
- The switch E-Block should be connected to Port D of node 1;
- The LED E-Block should be connected to Port D of node 2;
- The CAN, LCD and Switch E-Blocks all need to be connected to the +V on their associated EB006 via single core wire.
- The CANH signals should be linked together
- The CANL signals should be linked together
- The EB006 Ground signals should be linked together



Switch Settings—XTAL & Fast

Before you can begin you must install the driver for the EB006 using the ELSAM CD or by visiting the Matrix TSL website: http://www.matrixtsl.com

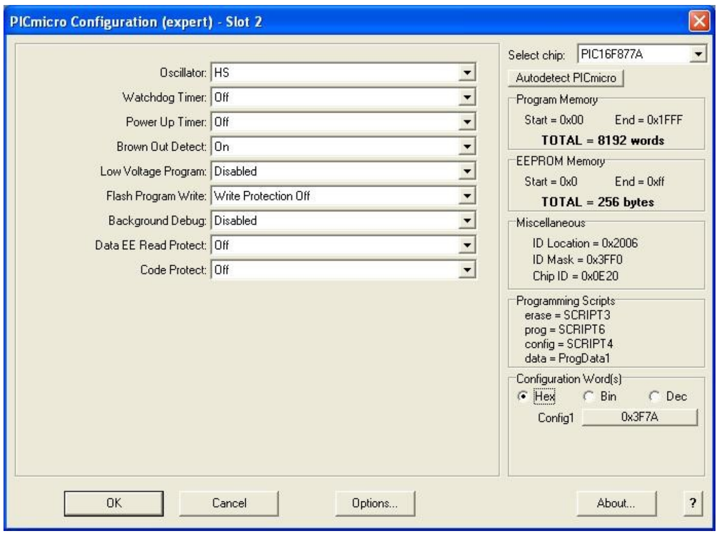


PICmicro Configuration (simple) - Slot 2

Select chip: PIC16F877A

Oscillator: RC [✓ XTAL] Internal

Watchdog timer: On [✓ Off]



PICmicro Configuration (expert) - Slot 2

Oscillator:	HS
Watchdog Timer:	Off
Power Up Timer:	Off
Brown Out Detect:	On
Low Voltage Program:	Disabled
Flash Program Write:	Write Protection Off
Background Debug:	Disabled
Data EE Read Protect:	Off
Code Protect:	Off

Select chip: PIC16F877A

Program Memory
Start = 0x00 End = 0x1FFF
TOTAL = 8192 words

EEPROM Memory
Start = 0x0 End = 0xff
TOTAL = 256 bytes

Miscellaneous
ID Location = 0x2006
ID Mask = 0x3FF0
Chip ID = 0x0E20

Programming Scripts
erase = SCRIPT3
prog = SCRIPT6
config = SCRIPT4
data = ProgData1

Configuration Word(s)
[✓ Hex] Bin Dec
Config1 0x3F7A

The system can be tested by compiling and sending one of the example programs to the hardware. This is done by opening one of the example files in Flowcode and then clicking the compile to chip button.

The example Flowcode files contain a correct configuration so you will not have to modify the configuration to allow them to run on the hardware.

Any program you create from scratch will have to be configured as shown on the left. The configuration is specified by clicking the Chip -> Configure menu icon Flowcode and then if you need more options click the switch to expert config screen button.

Example file 1 is a good test file as it is very basic and simply checks for communications between the CAN nodes. This allows for a quick and easy check to ensure that all of the nodes are working as they should.

Can Component

EB984-80-1

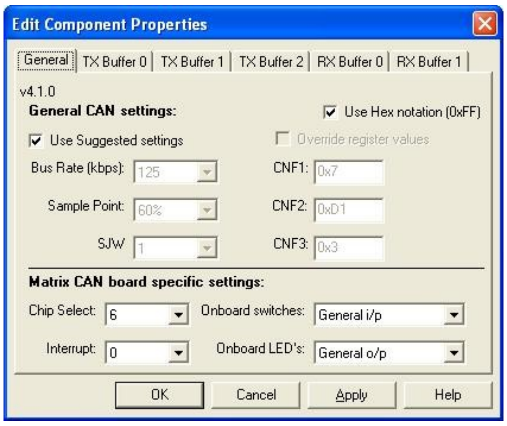
E-Blocks Easy CAN BUS Bundle

The Flowcode CAN component can be added to your program by finding the component in the Comms section of the Flowcode component toolbar.

The Flowcode CAN components are as follows:
CAN – Standard CAN message Identifiers
CAN2 – Extended CAN message Identifiers

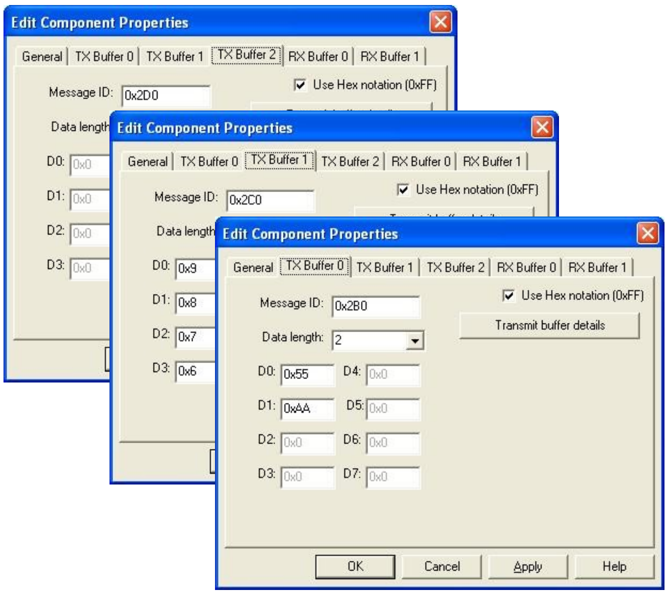
CAN component icon.

Once the component has been added to your program you will find that the CAN component icon has been added to your Panel.



v4.1.0
General CAN settings:
☑ Use Hex notation (0xFF)
☑ Use Suggested settings
☐ Override register values
Bus Rate (kbps): 125
Sample Point: 60%
SJW: 1
CNF1: 0x7
CNF2: 0xD1
CNF3: 0x3
Matrix CAN board specific settings:
Chip Select: 6
Interrupt: 0
Onboard switches: General i/p
Onboard LED's: General o/p

The main component properties screen is laid out as shown in the image on the left. This property page is responsible for configuring the CAN bus speeds and the pin connections for the chip select and interrupt data lines.

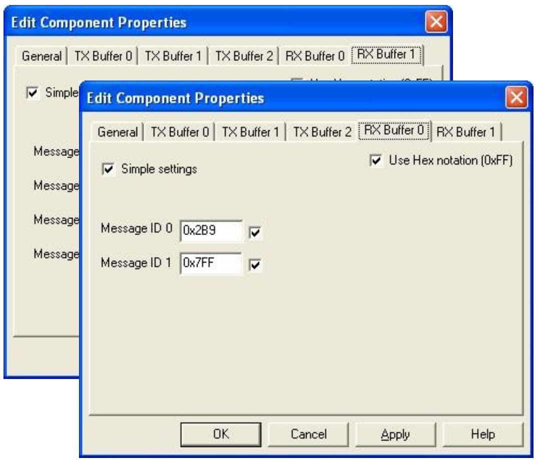


TX Buffer 2: ☑ Use Hex notation (0xFF)
Message ID: 0x2D0
D0: 0x0
D1: 0x0
D2: 0x0
D3: 0x0

TX Buffer 1: ☑ Use Hex notation (0xFF)
Message ID: 0x2C0
D0: 0x9
D1: 0x8
D2: 0x7
D3: 0x6

TX Buffer 0: ☑ Use Hex notation (0xFF)
Message ID: 0x2B0
Data length: 2
D0: 0x55
D1: 0xAA
D2: 0x0
D3: 0x0
D4: 0x0
D5: 0x0
D6: 0x0
D7: 0x0

There are three property pages which relate to transmit buffers. This allows you to predefine up to 3 outgoing messages complete with data values. The transmit buffer identifiers and data bytes can also be edited by the Flowcode program by using the CAN component macros.



RX Buffer 0:
☑ Simple settings
☑ Use Hex notation (0xFF)
Message ID 0: 0x2B9 ☑
Message ID 1: 0x7FF ☑

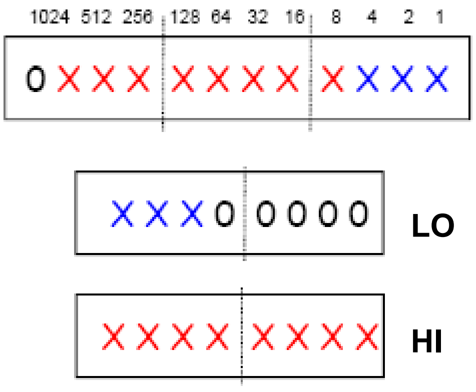
There are also two property pages which relate to receive buffers. These allow you to define combinations of identifier masks and filters to control the messages that the buffers can receive.

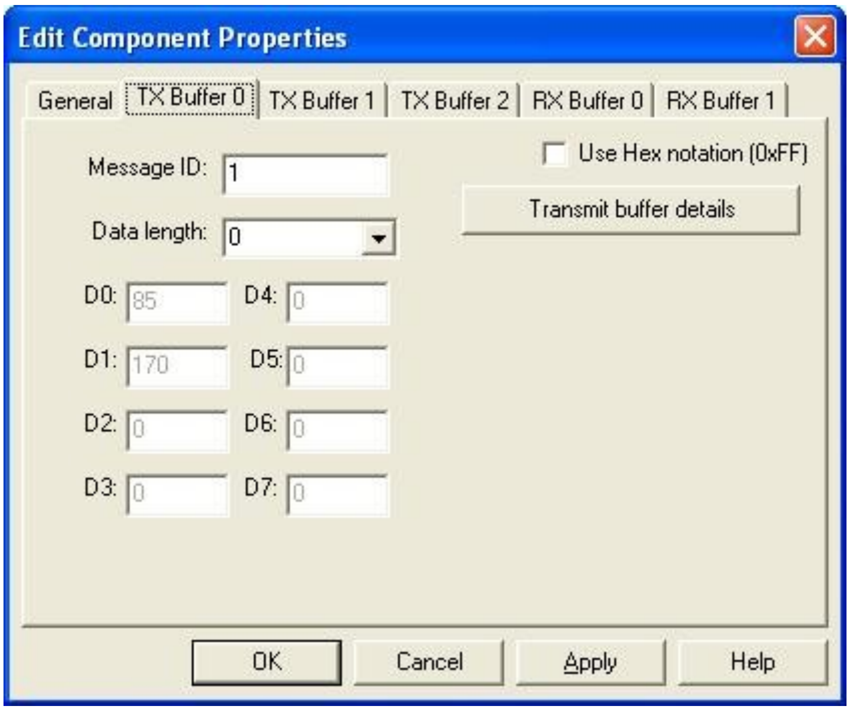
Further help on the CAN component can be found by selecting the component on the Flowcode Panel and then clicking the help button in the properties toolbar.

Example 1

Example 1 shows how to send and receive a message identifier complete with a single data byte. The node with the LCD and Switch E-Blocks will send out a CAN message every second containing the value of the switches. The receiving node will then forward any incoming data to the LEDs.

Each CAN message consists of an identifier and up to eight data bytes. The CAN identifier is made up of eleven bits to represent identifiers 0 - 2047. The eleven bit identifier shown on the right is made up of a high byte and a low byte. For example message identifier 1 would have a low id value of 0x20 and a high id value of 0.




Edit Component Properties

General | TX Buffer 0 | TX Buffer 1 | TX Buffer 2 | RX Buffer 0 | RX Buffer 1

Message ID: 1

Use Hex notation (0xFF)

Transmit buffer details

Data length: 0

D0: 85 D4: 0

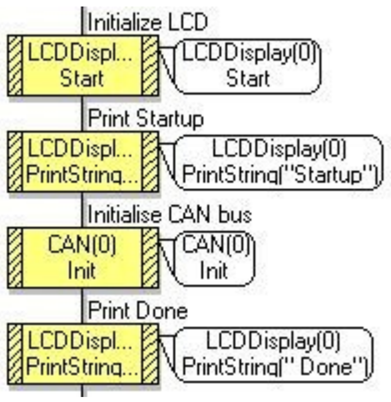
D1: 170 D5: 0

D2: 0 D6: 0

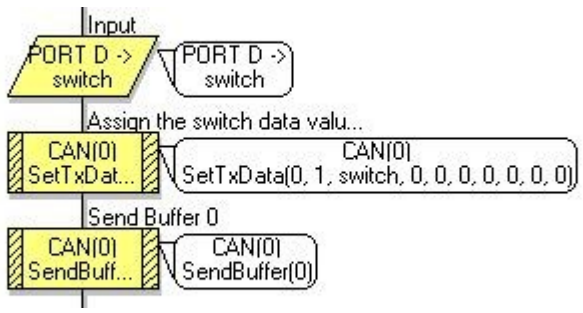
D3: 0 D7: 0

OK | Cancel | Apply | Help

In the example send program the transmit buffer is setup to use the identifier 1.

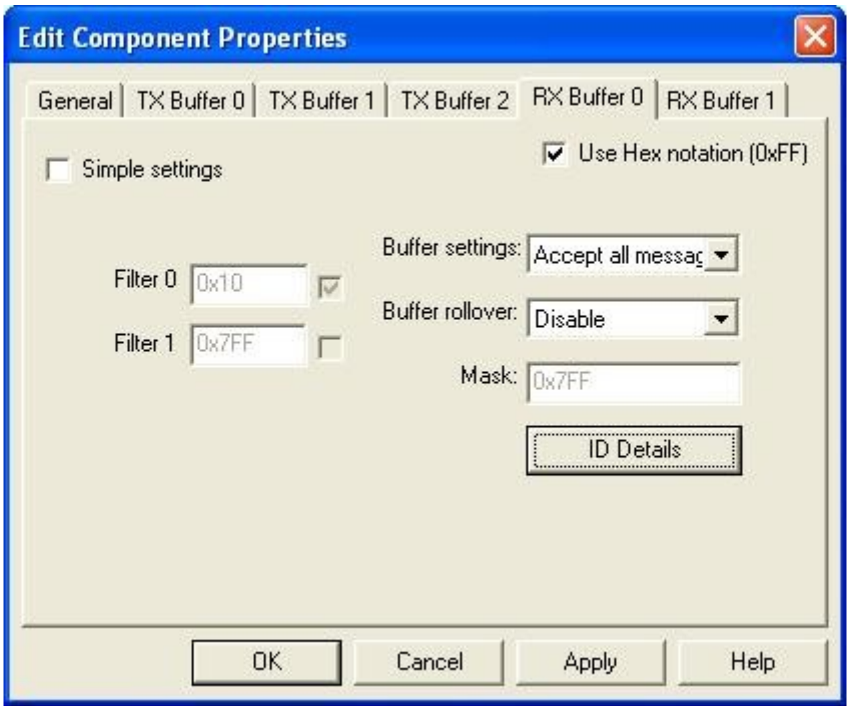

Initialize LCD
LCDDispl... Start
LCDDisplay(0) Start
Print Startup
LCDDispl... PrintString...
LCDDisplay(0) PrintString("Startup")
Initialise CAN bus
CAN(0) Init
CAN(0) Init
Print Done
LCDDispl... PrintString...
LCDDisplay(0) PrintString(" Done")

To initialise the CAN bus all we have to do is call the CAN component Init function using a component macro icon. If you are having a problem where the program seems to hang here then you will need to check that your connections are correct and that the E-Block CAN board is powered.



Data is assigned to the buffer by calling the SetTxData component macro. The buffer is then sent over the CAN bus by calling the send buffer command.

Example 1


Edit Component Properties

General | TX Buffer 0 | TX Buffer 1 | TX Buffer 2 | RX Buffer 0 | RX Buffer 1

☐ Simple settings

☑ Use Hex notation (0xFF)

Filter 0: 0x10 ☑

Filter 1: 0x7FF ☐

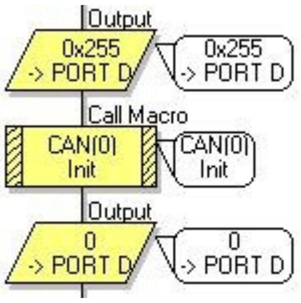
Buffer settings: Accept all messag

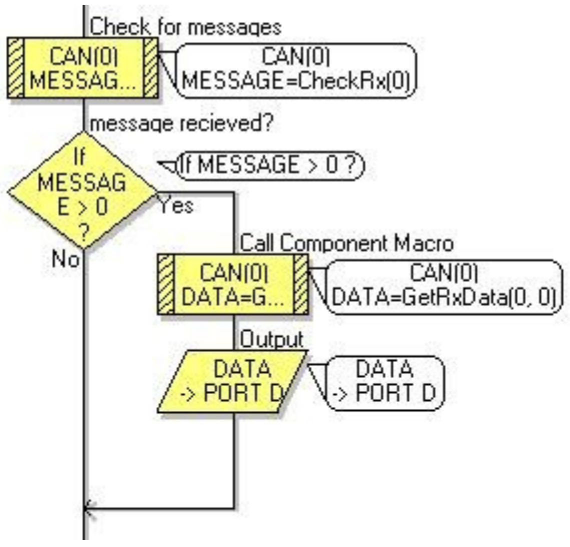
Buffer rollover: Disable

Mask: 0x7FF

For node 2 we configure receive buffer 0 to accept all message identifiers by un-ticking the simple setting option and changing the buffer settings dropdown menu item.

This time to detect a correct initialisation we are lighting up all of the LEDs by outputting the value 255 (8 - 1's in binary). When the initialisation function has finished we are outputting the value 0 to switch off all of the LEDs.





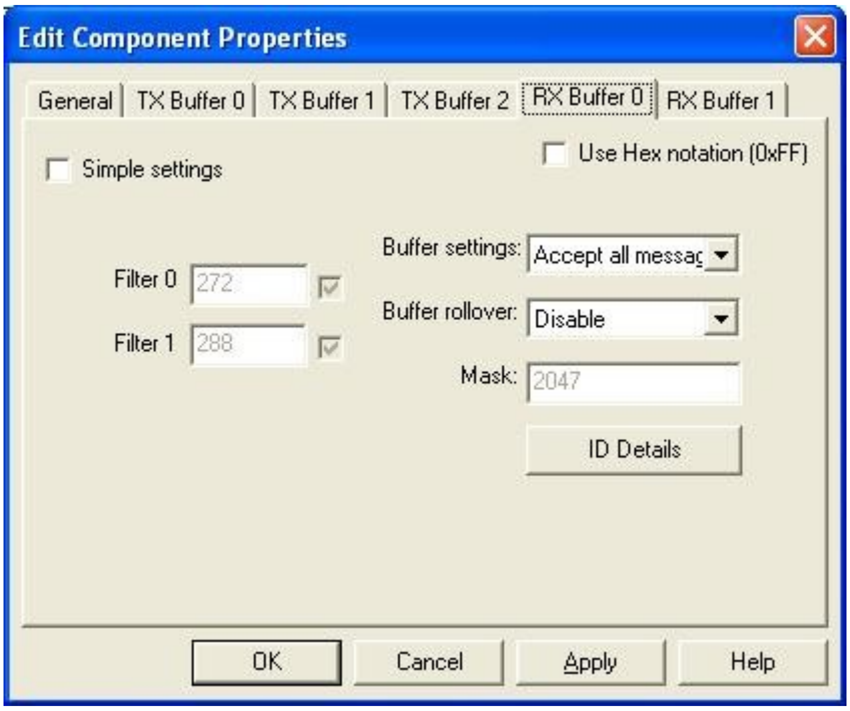
We then check for any incoming messages by calling the CheckRx component macro. Note that the CheckRx function will only see messages that get past the buffer mask and filter. In this example we have set the receive buffer to accept all messages so we should not have this problem. We then use the GetRxID lo and high functions to read the two bytes of the message ID. The GetRxDataCount component macro is used to read how many data bytes are in the message and the GetRxData component macro is used to retrieve a data byte from the message.

You should now have a working system where you can press the switches on one node and see the appropriate LED light on the other node.

Example 2

Example 2 adds to the functionality shown in example 1 by adding a detection routine to the programs. Node 1 now issues a message with an ID of 1 after starting up. Node 2 listens for messages as before but now decides what to do based on the message identifier that is received. If ID 1 is received then node 2 replies with an ID of 2. This tells node 1 that the message has been received and acknowledged. Node 1 then sends out a message ID of 3 to control the switch to LED functionality. Checking if a node is present before sending data to that node allows the system to be much more reliable and also allows it to sense if things go wrong.

For the node 1 program we have modified the receive buffer 0 to accept all incoming messages, This allows the node scan acknowledge ID to be detected by the component.



Edit Component Properties

General | TX Buffer 0 | TX Buffer 1 | TX Buffer 2 | RX Buffer 0 | RX Buffer 1

Simple settings ☐

Use Hex notation (0xFF) ☐

Buffer settings: Accept all messag

Filter 0: 272 ✓

Filter 1: 288 ✓

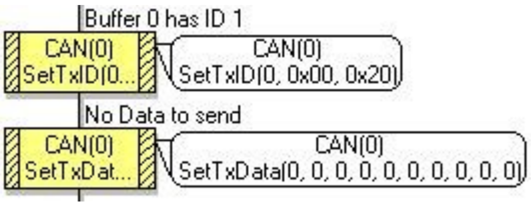
Buffer rollover: Disable

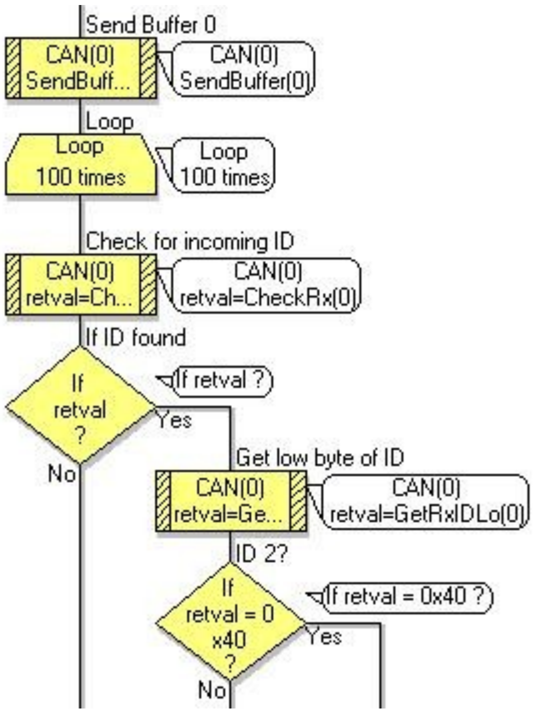
Mask: 2047

ID Details

OK | Cancel | Apply | Help

Here we modify the ID that is sent by transmit buffer 0. Note that the values used correspond to the ID format as shown on page 5.

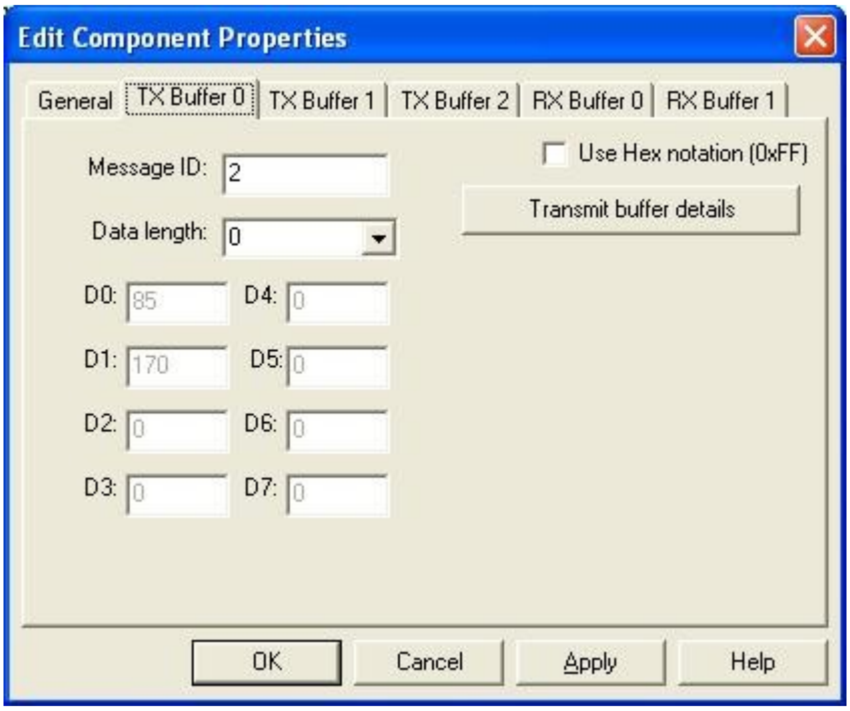

Buffer 0 has ID 1
CAN(0) SetTxID(0...
CAN(0) SetTxID(0, 0x00, 0x20)
No Data to send
CAN(0) SetTxDat...
CAN(0) SetTxData(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)



We sent the ID with the value of 1 and then perform 100 read attempts to allow the ID value of 2 to be received. If the ID is received and confirmed as being correct then we can drop out of the loop and start with the main program functionality.

Example 2

For the node 2 program we have modified the transmit buffer 0 to have a default ID of 2 and no data bytes. This means that unless we wish to change the ID or the data we can simply call the send macro to fire off this message.


Edit Component Properties

General | TX Buffer 0 | TX Buffer 1 | TX Buffer 2 | RX Buffer 0 | RX Buffer 1

Message ID: 2

Data length: 0

☐ Use Hex notation (0xFF)

Transmit buffer details

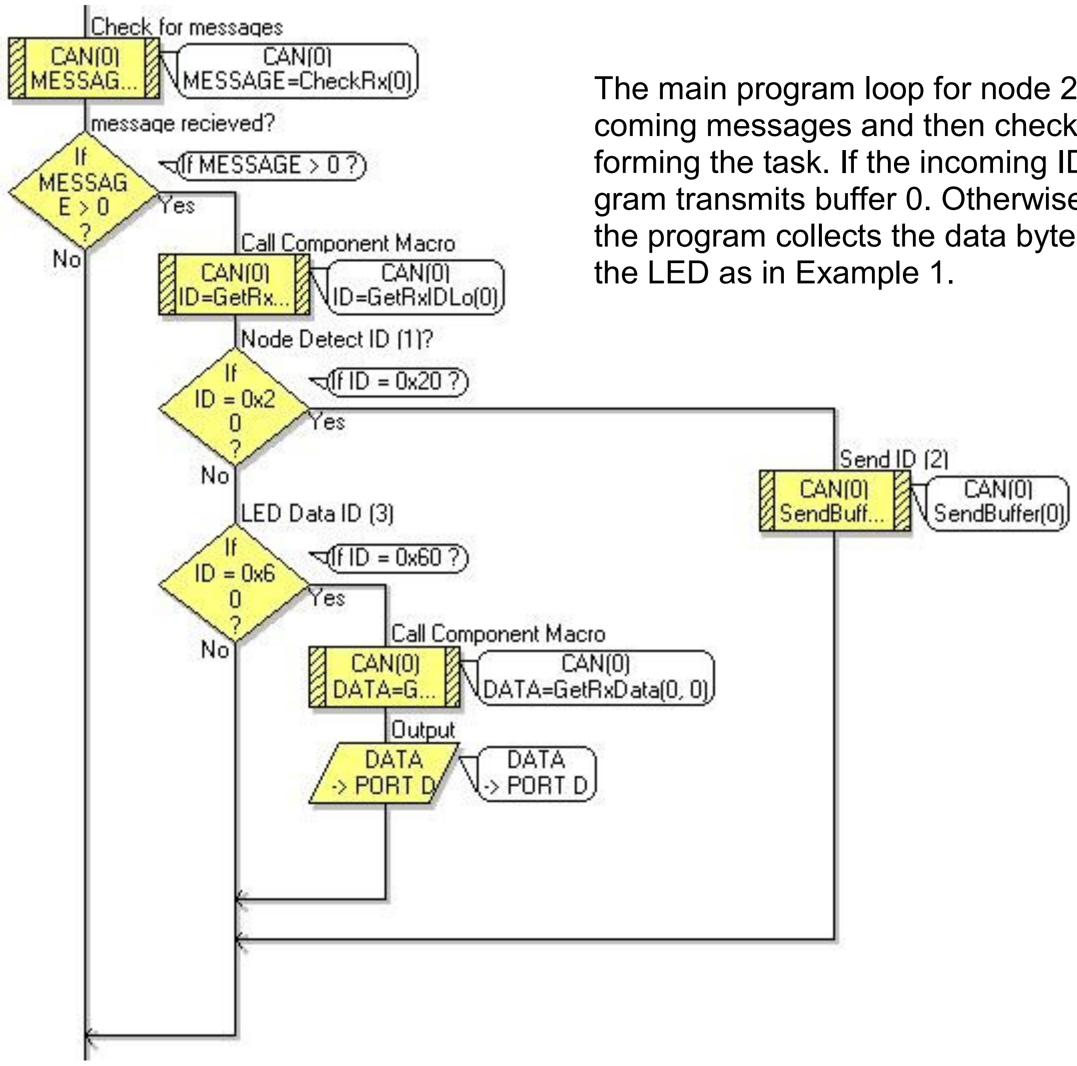
D0: 85 D4: 0

D1: 170 D5: 0

D2: 0 D6: 0

D3: 0 D7: 0

OK Cancel Apply Help

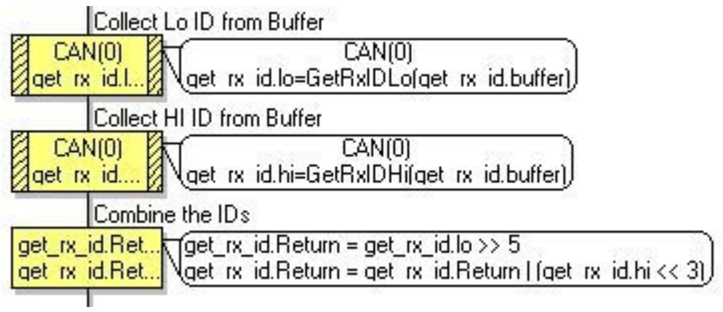


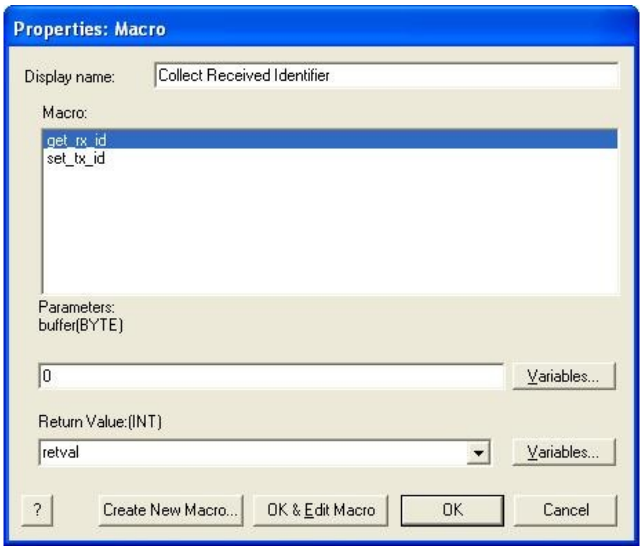
The main program loop for node 2 now checks for incoming messages and then checks the ID before performing the task. If the incoming ID is 1 then the program transmits buffer 0. Otherwise if the ID is 3 then the program collects the data byte and outputs it to the LED as in Example 1.

Example 3

Example 3 demonstrates two macros that can be used in your CAN bus applications to convert between the identifier values and the shifted lo and hi identifier values. The program is the same as example 2 but this time instead of having to work out what the lo and hi variables need to be for a certain identifier we can instead use the software macros to do the manipulations for us. The macros are available in the Examples folder for use with your own programs. Simply use the Macro -> Import menu to load the macros into your programs.

Get_RX_ID Macro


Collect Lo ID from Buffer
CAN(0)
get_rx_id.l...
CAN(0)
get_rx_id.lo=GetRxIDLo(get_rx_id.buffer)
Collect HI ID from Buffer
CAN(0)
get_rx_id....
CAN(0)
get_rx_id.hi=GetRxIDHi(get_rx_id.buffer)
Combine the IDs
get_rx_id.Ret...
get_rx_id.Ret...
get_rx_id.Return = get_rx_id.lo >> 5
get_rx_id.Return = get_rx_id.Return | (get_rx_id.hi << 3)


Properties: Macro

Display name: Collect Received Identifier

Macro:
get_rx_id
set_tx_id

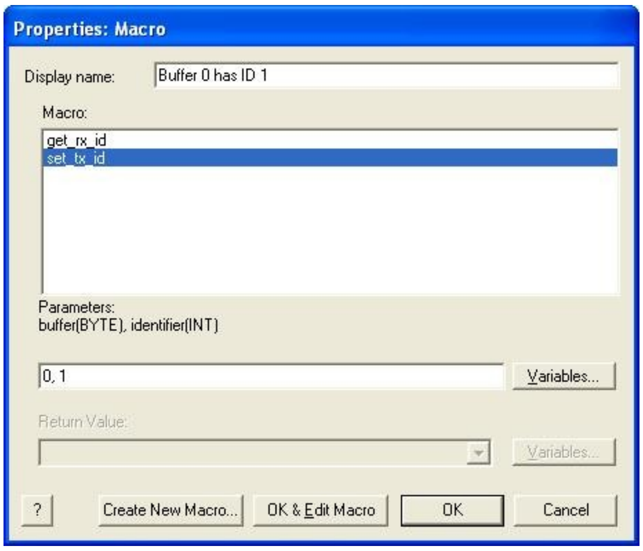
Parameters:
buffer(BYTE)

0

Return Value:(INT)

retval

The GET_RX_ID macro has a single parameter to specify which CAN receive buffer we want to read. The macro then reads the two identifier bytes as normal and then performs the conversion to combine the two bytes into a decimal integer value. Please note that this must only be called after a successful CheckRx component macro call to ensure that the data returned is correct and up to date. This allows you to receive an identifier value between 0-2047 without having to work out what the lo and hi byte values should be.


Properties: Macro

Display name: Buffer 0 has ID 1

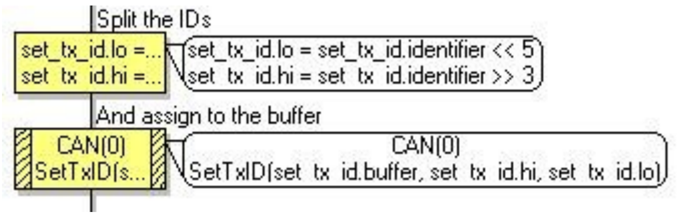
Macro:
get_rx_id
set_tx_id

Parameters:
buffer(BYTE), identifier(INT)

0, 1

Return Value:

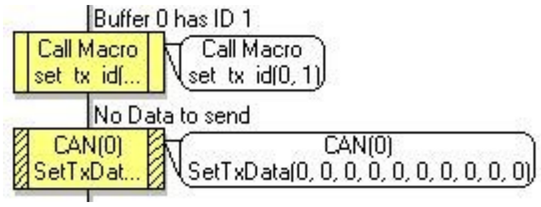
Set_TX_ID Macro

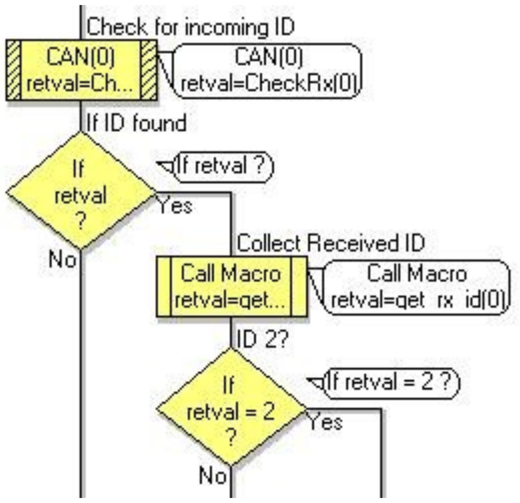

Split the IDs
set_tx_id.lo =...
set_tx_id.hi =...
set_tx_id.lo = set_tx_id.identifier << 5
set_tx_id.hi = set_tx_id.identifier >> 3
And assign to the buffer
CAN(0)
SetTxID(s...
CAN(0)
SetTxID(set_tx_id.buffer, set_tx_id.hi, set_tx_id.lo)

The SET_TX_ID macro has two parameters, one to specify the transmission buffer we will be using and the other to specify the identifier. The macro has the functionality of splitting the identifier up into the lo and hi bytes and then forwarding the modified values to the specified buffer. This allows you to enter an identifier between 0—2047 without having to work out what the lo and hi byte values should be.

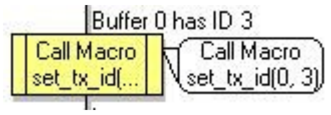
Example 3

Node 1

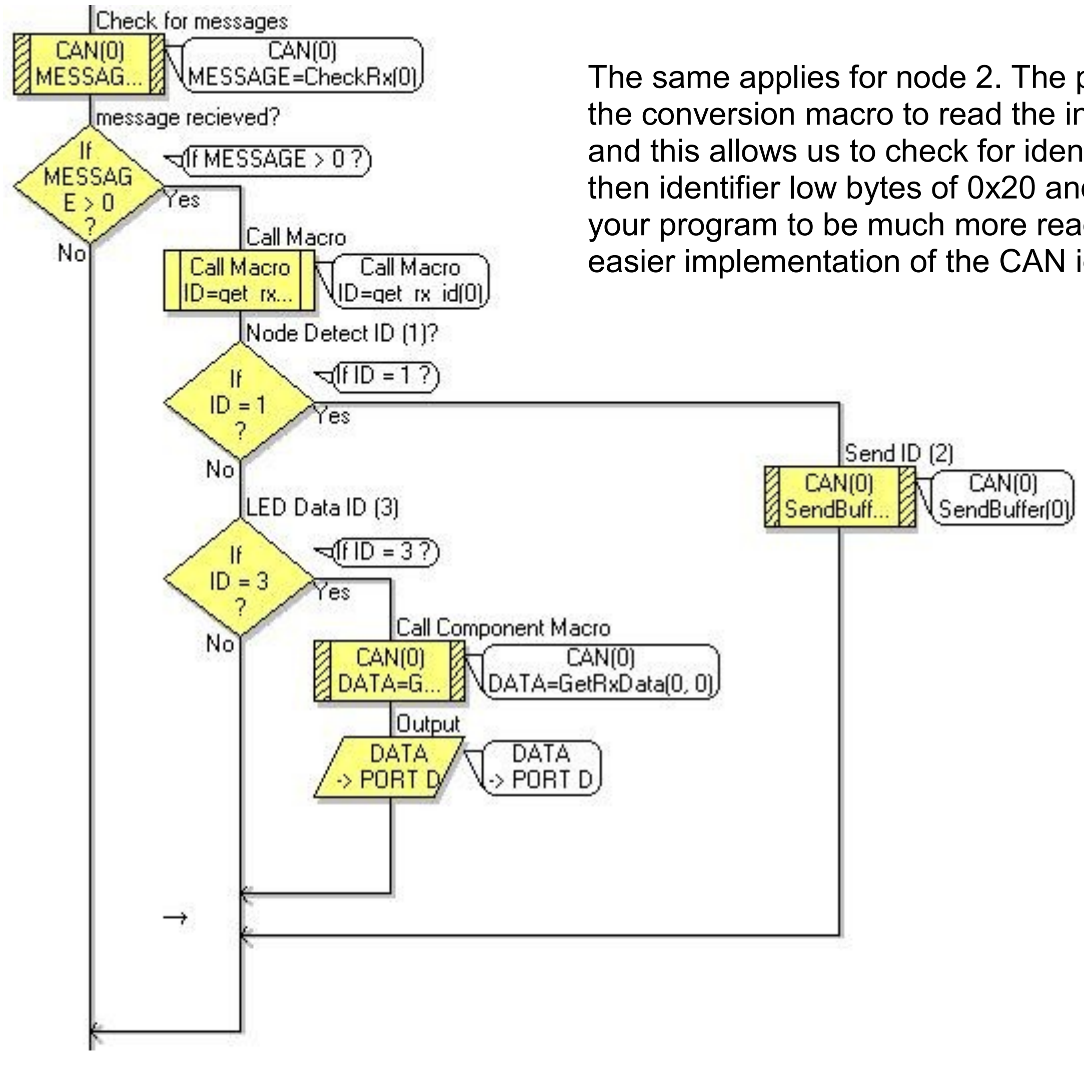




The node 1 program now changes so that we now set the transmission identifiers using the conversion macro. Note that we can now use the values 1 and 3. When waiting for the node acknowledge we can also check for the actual identifier value of 2 rather then 0x40 we were using previously.



Node 2



The same applies for node 2. The program now uses the conversion macro to read the incoming identifier and this allows us to check for identifier 1 or 3, rather then identifier low bytes of 0x20 and 0x60. This allows your program to be much more readable and allows for easier implementation of the CAN identifiers.

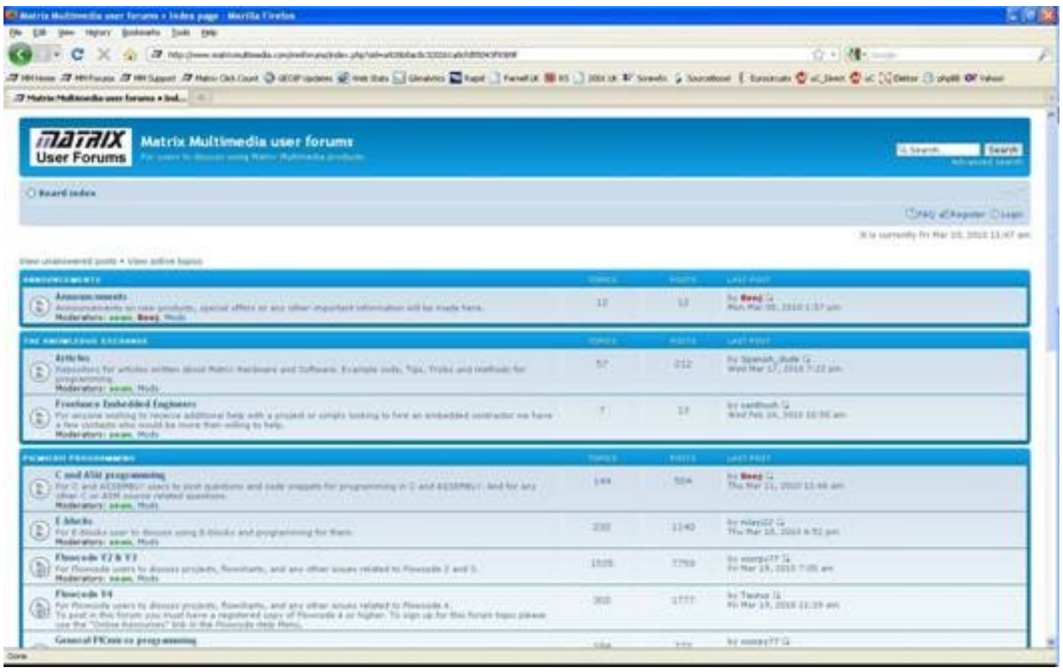
Troubleshooting

If you are having any problems getting up and running with any of the examples or any of the Flowcode components then the first port of call is to ensure you have your boards plugged together and wired correctly. As a rule of thumb any board with a screw terminal and a +V marking should be connected via a single core wire to the +V screw terminal on the corresponding Multiprogrammer.

If you do run into any problems then there is help and advice available from our online user forums located here:

http://www.matrixtsl.com/mmforums/

The Articles section contains quite a few examples, as well as hints and tips to aid in your applications



Matrix Multimedia user forums

Board index

Announcements

Articles

Freelance Embedded Engineers

C and ASM programming

E-blocks

Flowcode V2 & V3

Flowcode V4

General PICmicro programming

There is also an online video demonstrating Flowcode, available from the videos section of our website: http://www.matrixtsl.com/lc_videos.php

Other Products

EB984-80-1

E-Blocks Easy CAN BUS Bundle

Matrix TSL is a leading global technology company. Over the years we have developed a portfolio of award-winning products which have applications in Education, Industry and in the home.

Learning is at the heart of much of what our company does, and the philosophy of all Matrix learning products is based on 'learning by doing'. Each year Matrix spends around 25% of turnover on research and development to ensure that our learning and development resources are world class.

MIAC PLC

MIAC (Matrix Industrial Automotive Controller) is an industrial grade control unit which can be used to control a wide range of different electronic systems including sensing, monitoring and automotive. It has a number of applications in industry and learning.

Formula Flowcode

Formula Flowcode is a robot vehicle which is used to teach robotics, and to provide a platform for competing in robotics events.

ECIO

ECIO devices are powerful USB programmable microcontrollers with either 28 or 40 pin standard DIL (0.6") footprints. They are perfect for student use at home, project work and building fully integrated embedded systems.

FlowKit

The FlowKit allows for in circuit debugging directly from within Flowcode. This is the same ICD debugging feature that is included with our version 7 EB006 Multiprogrammer boards.

Flowcode + E-Block Technology bundles

Matrix TSL technology bundles are based on a combination of two of our most popular products, E-Blocks and Flowcode.

Other bundles in the range

- Easy Mobile Communications Pack
- Easy Zigbee Pack
- Easy RFID
- Easy Internet
- Easy CAN Bus
- Easy GPS
- Easy USB
- Build your own PC Interface
- Build your own PLC
- Build your own Data-logger