TABLE OF CONTENTS

1 Introduction – controlling fischertechnik models with ROBO Pro

You must have asked yourself at some time how it works when robots carry out their allotted tasks as if controlled by an invisible hand. But it's not just with actual robots, but in many other fields as well, that we encounter control and automation technology. Including fischertechnik. By the next chapter but one, we will be designing a little control program for an automatic garage door together, and in doing so we'll learn how control problems like this can be solved and tested with the help of ROBO Pro software for Windows. ROBO Pro is also very simple to operate. Control programs, or more precisely flow charts and later data flow charts, as we shall learn, can be created on the graphical user interface, almost exclusively using the mouse.

In order to be able to control your fischertechnik models through your PC, you will need, as well as the ROBO Pro control software, an Interface to connect the computer with the model. It transforms the software commands so that, for example, motors can be controlled and sensor signals can be processed. The ROBO TX Controller (item number 500995), the earlier ROBO Interface (item number 93293) and the Intelligent Interface (item number 30402) are available from fischertechnik. You can use any of these Interfaces with ROBO Pro. But ROBO Pro only supports the online mode of the Intelligent Interface. ROBO Pro no longer supports the very early parallel Interface (item number 30520).

A few words about the layout of this manual. It is divided into two parts. The first part, from Chapter 1 to Chapter 4, describes the basic procedure for programming with ROBO Pro. This gives you a lot of information and background knowledge about programming in general and about how to use the ROBO Pro software.

The second part consists of Chapters 5 to 7, and gives an introduction to the functions needed for more advanced programs.

Chapters 8 onwards are more for reference. So when you're familiar with the operation of ROBO Pro after reading the first part and you need very specific information, here is where you will find comprehensive explanations of the individual program elements.

> If you are already familiar with ROBO Pro and only want to find out what new features were added with the ROBO TX Controller, you should read only chapters 11 through 13 of the manual.

So let's go! You must already be itching to know what possibilities ROBO Pro gives you for programming your fischertechnik models. Have fun!

1.1 Installation of ROBO Pro

System requirements for installing ROBO Pro are:

- an IBM-compatible PC with Pentium II processor with a clock speed of at least 500 MHz, 64 MB RAM and ca. 40 MB free disk-space on the hard drive
- a monitor and a graphics card with a resolution of at least 1024x768 pixels. With CRT monitors the refresh rate should be at least 85 Hz to maintain a flicker-free image. TFT flat screens provide a flicker-free image at any refresh rate, so that the refresh rate is not critical with TFT flat screens.

- Microsoft Windows, Version Windows XP or Vista
- A free USB interface to connect the ROBO TX Controller. To connect the ROBO Interface you need a free USB interface or a free RS232 interface (COM1 to COM4).

First of all, of course, you must start the computer and wait until the operating system (Windows) has finished loading. The ROBO Interface should only be connected to the computer after successful installation. Insert the installation CD into the CD-ROM drive. The installation program on the CD will then be started automatically.

- In the first Welcome window of the installation program you push the **Next** button.
- The second window, **Important Notes**, contains important up-to-date notes about installing the program or about the program itself. Here too, you click on the **Next** button.
- The third window, **License Agreement**, displays the ROBO Pro licensing contract. You must click **Yes** to accept the agreement before you can proceed to the next window with **Next**.
- In the next window, **User Details**, please enter your name.
- The next window, **Installation Type**, allows you to choose between **Express Installation** and **Customized Installation**. With customized installation, you can choose to leave out individual components of the installation. If you are installing a new version of ROBO Pro over an older version, and you have modified some of the sample programs in the older version, you can exclude the sample programs from the customized installation. If you don't do this, the modified sample programs will be ***overwritten without warning***. If you select customized installation and press **Next**, an additional window, allowing you to select the components, will appear.
- In the **Target directory** window you can select the folder or directory path where you want the ROBO Pro program installed. This will normally be the path C:\Program Files\ROBOPro. However, you can also enter another directory.
- When you push the **Finish** button in the last window, the installation is performed. As soon as the installation is finished – this normally only takes a few seconds – the program announces successful installation. If there are problems, an error message is displayed, which should help you to solve the problem.

1.2 Installing the USB driver

This step is required if the ROBO TX Controller or the ROBO Interface is to be connected to the USB port. The ROBO Interface can also be connected to one of the serial ports COM1-COM4. The earlier Windows versions Windows 95 and Windows NT4.0 don't support USB ports. With Windows 95 or NT 4.0, the ROBO Interface can only be connected via the serial port. There is no need to install a driver in this case.

Important note for the installation under Windows 2000, XP and Vista:

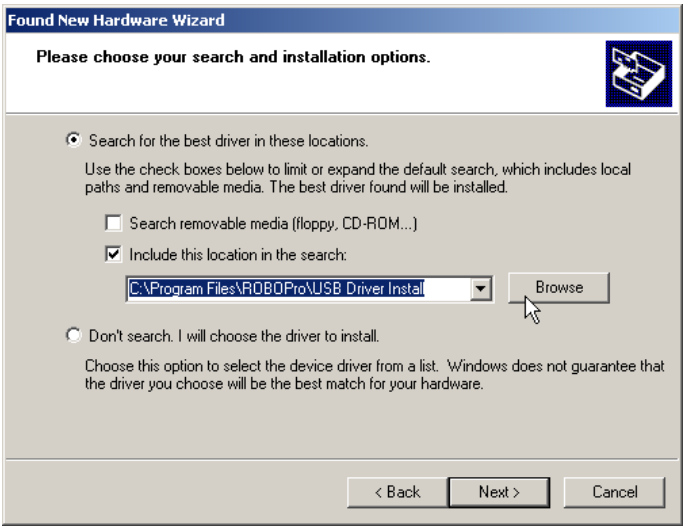
The USB driver can only be installed by a user with PC systems administrator privileges. Should the installation program advise you that you are not permitted to install the USB driver, you must either ask your system administrator to install your driver or install ROBO Pro without this driver.

In order to install the USB driver, you must first connect the ROBO TX Controller or the ROBO Interface with a USB cable to your computer and supply it with power. Windows recognizes automatically that the Interface is connected and displays the following window:

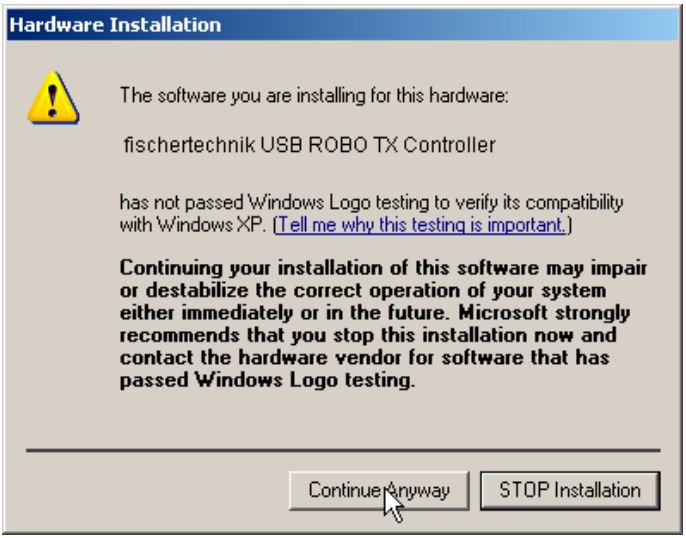
Depending on the operating system, the appearance of the window can be somewhat different from the illustration above!

Here you must select **Install software from a list or specific source** and press **Next**.

In the next window you deactivate **Search removable media** and activate **Also search following sources**. Then you click **Search** and select the sub-directory ***USB Driver Installation*** in the directory in which ROBO Pro is installed (the standard directory is C:\Program Files\ROBOPro\). For the ROBO TX Controller you first select the sub-directory ***TXController,*** for the ROBO Interface the sub-directory ***ROBOInterface,*** and then the sub-directory containing the driver for your operating system, for example ***WinXP***.



Under Windows XP, you may see the following message after pressing **Next**:



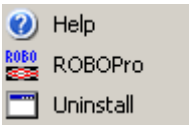
The USB driver is still being tested by Microsoft. Once testing is completed the driver will be approved by Microsoft, so that this notice no longer appears. In order to install the driver, press **Proceed with installation**.

Finally, the following message will appear:

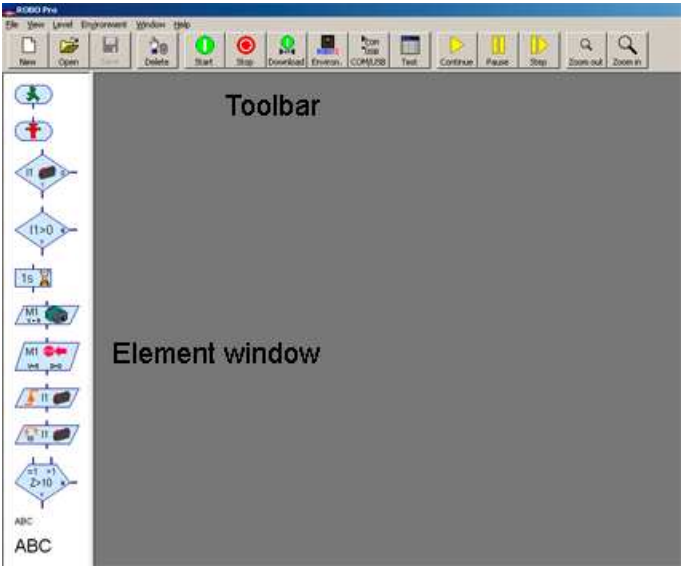
Press **Finish** to complete USB driver installation.

1.3 First Steps

Curious? Then simply start the program ROBO Pro. To do this, you click on the Start button on the task bar and then select **Programs** or **All programs** and **ROBO Pro**. In this folder of the Start menu you will find the following entries:



The Uninstall entry allows you to uninstall ROBO Pro. The Help entry opens the ROBO Pro Help file, and the ROBO Pro entry opens the ROBO Pro program. Now select the entry **ROBO Pro** to launch the program.

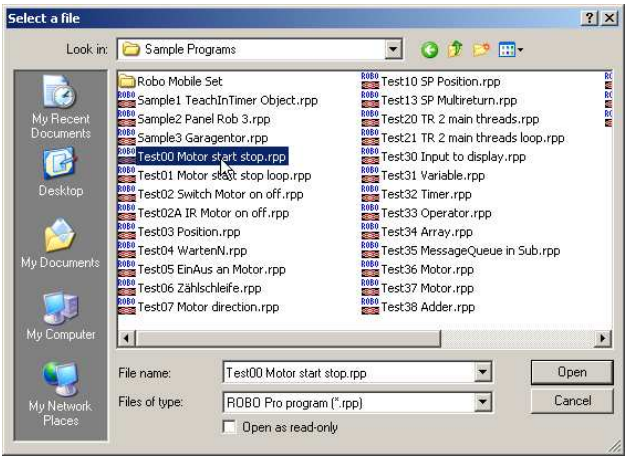


The window has a menu bar and toolbar with various operating buttons above as well as a window on the left-hand side with program elements. If you see two stacked windows in the left margin, ROBO Pro is not set on **Level 1**. To allow the functionality of ROBO Pro to match your growing knowledge, you can set ROBO Pro from Level 1 for beginners up to Level 5 for experts. Look in the **Level** menu to see whether there is a checkmark by **Level 1: Beginners**. If not, please switch to level 1.

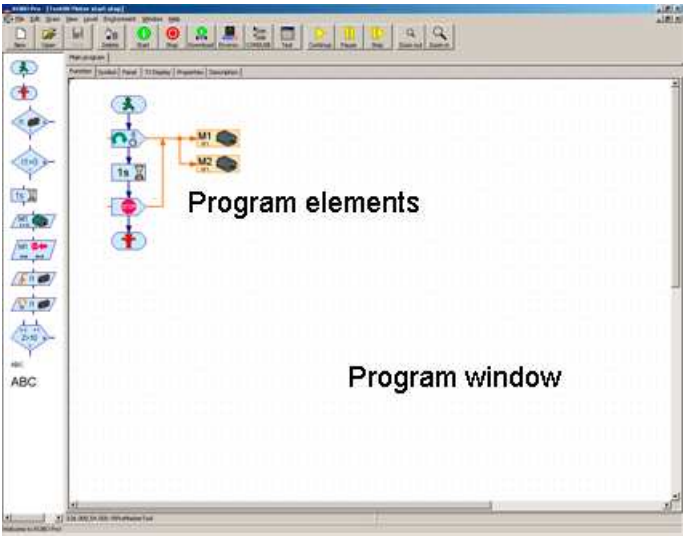
ROBO Pro is configured to use the ROBO TX Controller as interface. You can see this by the presence of the button ROBO TX in the toolbar. In *Chapter 11.2 Environment* you learn how you can switch to the earlier ROBO Interface and what you need to pay attention to.


Open

Now you may either create a new program file or open an already existing program file. We do not intend to create a new program file until Chapter 3, when we will write our first control program. To familiarize ourselves with the user interface, we shall open an already existing sample program. To do this, you click the entry ***Open*** in the ***File*** menu, or use the ***Open*** button in the toolbar. The sample files are found in the folder ***C:\Program Files\ROBOPro\Sample Programs***.



Open the file **\Level3\Motor start stop.rpp**:



Here you can see what a simple ROBO Pro program looks like. In programming, control-program flow charts are created in the program window using program elements from the element window. The finished flow charts can then be checked before being tested using a connected fischertechnik Interface. But not too fast: we shall learn programming step-by-step in the following chapters! Having thus gained your first impression of the user interface, you close the program file using the

Close command in the **File** menu. You can answer **No** to the question of whether you want to save the file.

2 A quick hardware test before programming

Clearly, the Interface must be connected to the PC for us to be able to test the programs we will later create. But, depending on the Interface used (ROBO TX Controller or ROBO–Interface), appropriate interface connection settings must also be made and tested. We sill do this in the coming chapter.

2.1 Connecting the Interface to the PC

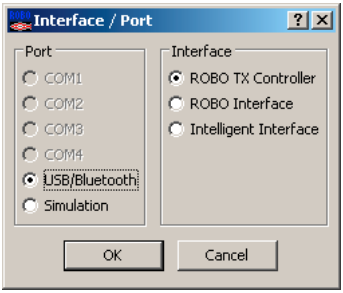
This should not be a great problem. The connecting cable supplied with the Interface is connected to the Interface and to a port on the PC:

- With the ROBO TX Controller a USB port can be used
- With the ROBO Interface (item number 93293) either a USB port or one of the serial ports COM1 to COM4 can be used.

The connections for these ports are normally found on the back of the computer. The exact placement of the various connections is described precisely in the user manual of your PC; please look it up there. USB connections are also often found on the front of a PC. Don't forget to give the Interface a power supply (mains unit or battery). The individual connections of the Interface are described in detail in the user manual of the respective equipment.

2.2 Getting the right connection – Interface settings

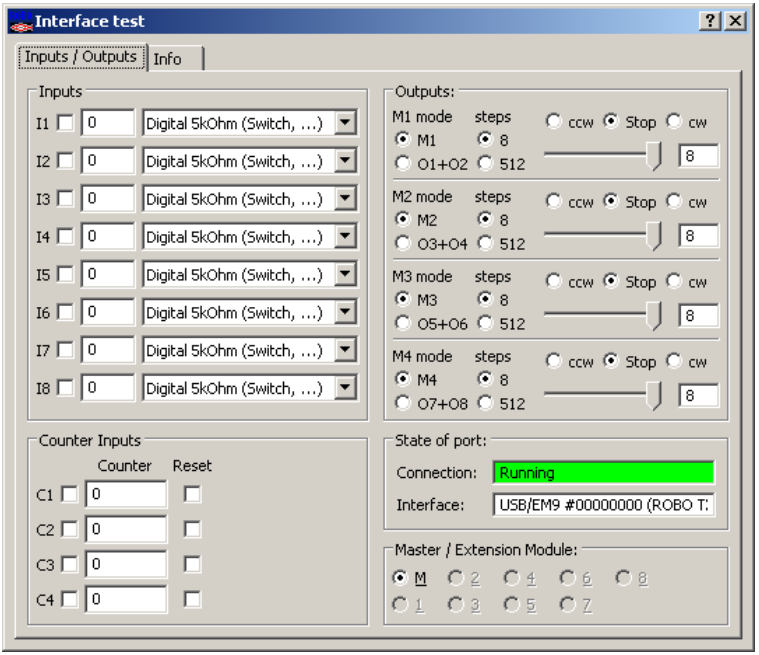
For the connection between the Interface and the PC to function correctly, ROBO Pro must be configured for the Interface currently in use. To do this, start ROBO Pro using the **ROBO Pro** entry on the Start menu under ***Programs*** or ***All programs*** and ***ROBO Pro***. Then push the **COM/USB** on the toolbar. The following window will appear:



Here you can select the port as well as the Interface type.


Test

Once you have selected the appropriate settings, close the window with **OK**. Now open the Interface test window with the **Test** button on the toolbar.



It shows the inputs and outputs available on the Interface. The green bar in the lower left of the window displays the connection status of the PC to the Interface:

- **Connection: Running** confirms correct connection to the Interface
- **Connection: Stopped** indicates that the connection has not been correctly set up and the PC was unable to establish a connection to the Interface. In this case, the bar will appear red.

To be able to change the Interface or connection settings, you must close the Test window (with the **X** in the upper right) and select another port or another Interface type as previously described, via the **COM/USB** button in the toolbar.

If you have been able to set up the connection between PC and Interface as described and the green bar appears, you will be relieved to know you can skip the next section.

If not, perhaps the tips in the next section can help you out.

2.3 Wrong connection: no connection to the Interface!?

If you get the message **Stopped** with your interface despite having correctly set the port (see above), you should check the following points. For this purpose, you may need to get advice from a computer expert:

- ***Power supply:***
  Does the Interface have an appropriate power supply? If you are using disposable or rechargeable batteries as power supply, the possibility arises that these are flat and no longer

supply sufficient voltage. If the battery voltage falls below 6 V, the ROBO TX Controller's processor may stop working. In this case the display will stop showing any information. If the voltage is too low, you must replace or, where appropriate, recharge the batteries, or, if possible, test the Interface with a mains power supply.

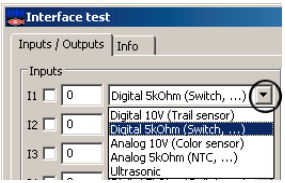
- ***Has the USB driver been installed correctly?***
  You can find this out by checking in the Device Manager in the Windows Control Panel whether the entry fischertechnik USB ROBO TX Controller appears under connections (COM and LPT) and functions properly. Should this entry not appear, install the USB driver again. If an error appears, uninstall the driver (click on the respective entry with your right mouse button) and install it once again.
- Is there a conflict with another device driver on the same port (e.g. a modem)? This driver may need to be deactivated (see Windows or device handbooks).
- If you still can't establish a connection to the Interface, then probably either the Interface or the connection cable is faulty. In this case, you should consult fischertechnik Service (Address: see menu: **"?"** / **Information about**).

2.4 Is everything working – the Interface test


Test

Once the connection has been correctly set up, you can use the Interface test to test the ROBO TX Controller and the models connected to it. The test window displays the various inputs and outputs of the Controller:

- ***Universal inputs I1—I8***
  I1—I8 are the universal inputs of the ROBO TX Controller. This is where different types of sensors can be connected. There are digital and analog sensors. You set the universal inputs depending on the type of sensor you would like to connect.



  - ***Digital sensors*** can only assume the states 0 and 1, or Yes and No. By default, both universal inputs are set to the input type Digital 5kOhm. Switches (mini pushbutton-switches), as well as phototransistors (light sensors) or reed-switches (magnetic sensors), can be connected to these digital inputs.
    You can check the functioning of these inputs by connecting a mini-sensor (item number 37783) to the Interface, e.g. to I1 (use contacts 1 and 3 on the switch). As soon as you press the button, a check-mark appears in the display of I1. If you have connected the switch the other way around (contacts 1 and 2), the check-mark will appear straight away and disappear when you press the button.
- The setting **Digital 10V** is used fort he infrared trail sensor.
- The setting **Analog 10V** can be used for the color sensor or to measure voltages between 0 and 10V such as the supply voltage of the battery pack. The voltage is displayed in mV (millivolt).
- **Analog 5kOhm** is used for the NTC resistor to measure temperatures and for the photoresistor to measure light. Here the reading is displayed Ohm (Ω = electrical resistance).
- The setting **Distance** is used for the ultrasound distance sensor (for the ROBO TX Controller only the version TX of the distance sensor with 3 pin connection cable, item number 133009, can be used).

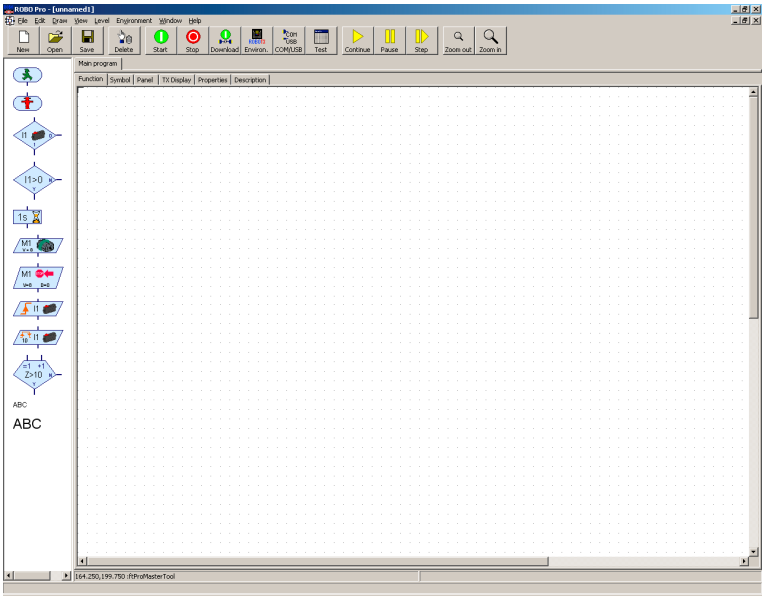
- ***Counter inputs C1-C4***
  These inputs allow you to count fast pulses with frequencies of up to 1000 pulses per second. You can also use them as digital inputs for buttons (not suitable for the trail sensor). If you connect a button to this input, every push of the button (=pulse) will increase the value of the counter by 1. This allows you, for example, to let a robot travel a specific distance.
- ***Motor outputs M1—M4***
  M1 – M4 are the outputs from the Interface. This is where what are called actuators are connected. These can be, e.g., motors, electromagnets or lamps. The 4 motor outputs can be controlled in speed and in direction. Speed is controlled using the slide control. You can choose between a coarse resolution with 8 different steps of speed or a fine resolution with 512 steps. The program elements in levels 1 and 2 only use the coarse resolution, but starting with level 3, there are elements which allow you to use the fine resolution. The speed is displayed next to the slider control as a number. If you would like to test an output, you connect a motor to an output, e.g. M1.
- ***Lamp outputs O1—O8***
  Each motor output can alternatively be used as a pair of individual outputs. These can be used to control not only lamps, but also motors which only need to move in one direction (e.g. for a conveyor belt). If you would like to test one of these outputs, you connect one lamp contact to the output, e.g. O1. You connect the other lamp contact with one of the ground sockets of the R (⊥).
- ***Extension modules***
  The ROBO TX Controller connected to the PC via the USB port (=master) can take up to 8 additional ROBO TX Controller as extensions (see manual ROBO TX Controller). These buttons allow you to select which of the connected devices you would like to access with the test window.

3 Level 1: Your first control program

After testing the hardware, that is the Interface and the switches and motors connected to it, in Chapter 1, we'll now get down to programming. But what does "programming" actually mean? Well, just imagine that, for example, a robot is connected to our Interface. But this robot is so stupid that it can't do anything on its own. Luckily, we're a bit smarter than that. We can tell the robot exactly what to do. How? Well, what happened in the last chapter when we used the mouse button to set the motor output M1 on "left"? Right, we switched the motor on. If, for example, this motor were to drive the gripping claw of our robot, we would have done nothing else than to say to the robot: "Grip the object!" But now we don't want to initiate every step by hand; rather the robot should do this automatically. To achieve this, we must store the individual steps to be carried out, so that the robot can work through them one after another, i.e., we must create a program, which will control the robot on our behalf. Logically enough, the technical term for this is a control program.

3.1 Creating a new program

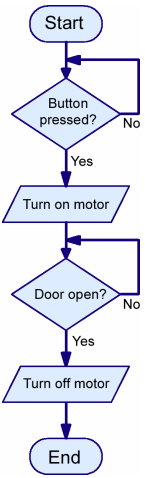
The ROBO Pro software gives us a great tool to design these control programs and to test them with the aid of a connected Interface. Don't worry: we're not about to program the robot straight away. We shall content ourselves initially with simple control tasks. To do this we must create a new program. In the toolbar you will find the entry **New**. If you left-click on it with your mouse, a new, empty program is created.



Now you see a large white drawing surface, in which you will enter your first program. If you see two stacked windows in the left margin, please switch to **Level 1: Beginners** in the **Level** menu.

3.2 The elements of a control program

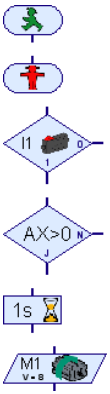
Now we can set about creating our first control program. We shall do this on the basis of a concrete example:



Functional description:

Imagine a garage door that can be opened automatically. Maybe you've even got one at home! You arrive at the garage in your car and, with the push of a button on the transmitter, the door, driven by a motor, is opened. The motor must keep running until the garage door is completely opened.

Words are a rather cumbersome and not very graphic way to describe a control program. So what we call ***flow charts*** are used to represent the sequence of actions to be performed and the conditions that need to be fulfilled for these actions. In the case of our control system, the condition for the action "switching on motor" is that the button be pressed. It is easy to read one of these flow charts: just follow the arrows step-by-step! These show exactly how the control system works – the individual steps can only be carried out in the order given by the arrows, never in any other way. Otherwise it wouldn't be worth going to all the trouble, would it?



Using our ROBO Pro software, we can now draw precisely this flow chart and in so doing create the ***control program*** for the connected hardware (Interface, motors, switches, etc.). The software does the rest, which, as it happens, is just the way it is with large industrial applications too! So we can concentrate fully on the creation of the flow chart.

You put the flow chart together from program elements. Another new concept? Don't worry! In ROBO Pro the individual elements that are put together to form a flow chart are called program elements. The action "switch on motor" means just that: the Interface should actually switch on the motor that is connected to it! You will find the available program elements in the element window on the left-hand side.

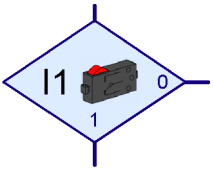
3.3 Inserting, moving and modifying program elements

Now it's a matter of creating a flow chart for our garage door control system from the program elements contained in the element window. All available program elements can be fetched from the element window and inserted in the program window.

Inserting program elements.

You move the mouse onto the symbol for the desired program element and left-click on it once. Then you move the mouse into the program window (that's the large white area) and click once again. You can also drag the program element into the Program window while holding down the mouse button. A program always begins with a Start element. The Start element is the rounded element with the little green GO man. It would be best to try this out straight away with this program element: Left-click once on the Start element in the element window, move the mouse up into the program window and once there left-click once more.

The next element in the program flow chart queries an input and then branches to one path or another depending on its state. In the element window, click on the element depicted right and then move the mouse below the previously inserted Start element. If the upper input of the Branch element is one or two grid points below the exit of the Start element, a connecting line will appear in the program window. If you left-click again, the Branch element is inserted and automatically connected with the Start element.



Moving program elements and groups

A program element can be moved to the desired position after insertion while holding down the left mouse button. If you want to move several elements as a group, you can start by drawing a frame around the elements while holding down the left mouse button. To do this you have to left-click in an ***empty*** zone, keep the button pressed and use the mouse to draw a rectangle containing the desired elements. The elements in the rectangle are now displayed with a red border. If you now move one of the red elements with the left mouse button, all the red elements are moved. You can also mark individual elements red by left clicking on them while holding down the shift key (i.e. the upper/lower case key). If you left click in an empty zone, all the red-marked elements will be displayed normally again.

Copying program elements and groups

Copying program elements and groups can be done in two ways. You can proceed exactly as for moving, except that you press the **CTRL** key on the keyboard before moving the elements. In this way the elements are not moved, but copied. However, with this function you can only copy elements within a program. If you want to copy elements from one program to another, you can use the Windows ***clipboard***. First select some elements, as described in the previous section in the case of moving elements. If you now hit **CTRL+C** on the keyboard or click on **Copy** on the **Edit** menu, all the selected elements will be copied onto the Windows clipboard. Now you can change over to another program and re-insert the elements there with **CTRL+V** or **Edit / Paste**. Once elements are copied, you can also paste them in several times. If you want to move elements from one program to another, you can use **CTRL+X** or **Edit / Cut** function at the beginning instead of **CTRL+C** or **Edit / Copy**.

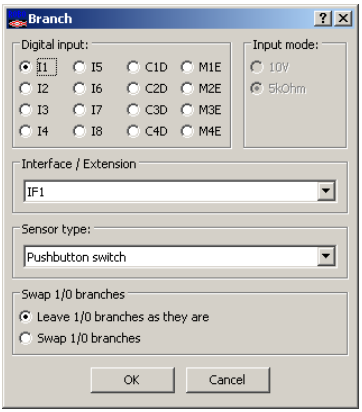
Deleting elements and Undo function


Delete

It is quite simple to delete elements. You can delete all the elements marked in red (see previous section) by pressing the “delete” key (**Del**) on the keyboard. You can also delete individual elements with the Delete function. To do this, first click on the button in the toolbar like the one illustrated and then on the element you want to delete. Try it out now. Then you can redraw the deleted element. But you can also retrieve the deleted element using the **Undo** function in the **Edit** menu. By using this menu item you can undo any changes to the program.

Editing program element properties

If you ***right*** click on a program element in the program window, there will appear a dialog window, in which you can change the element's properties. The Properties window for a Branch element is illustrated on the right.

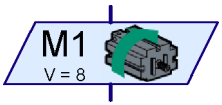


- Buttons **I1** to **I8** allow you to enter which of the Interface's inputs is to be queried. The inputs C1D-C4D correspond to counter inputs if you use them as digital inputs. We will deal with the inputs M1E-M4E later.
- The additional inputs C1D-C4D and M1E-M4E are covered in the reference section in 8.1.3 *Digital Branch* on page 75.
- The selection **Interface / Extension** is not explained until Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Sensor type** you can select the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed-contact switches. Selecting the sensor automatically sets the required input type for the universal inputs I1-I8 of the ROBO TX Controller.
- Under **Interchange 1/0 connections** you can interchange the positions of the 1 and 0 exits of the Branch element. Normally the 1 exit is below and the 0 exit is on the right. But sometimes it's more practical to have the 1 exit on the right. Press on **Interchange 1/0 connections** and the 1 and 0 connections will be changed over as soon as you close the window with **OK**.

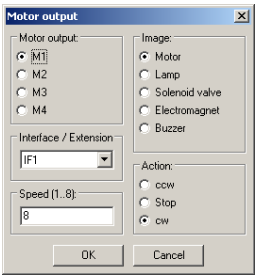
Hint: If you connect a mini-sensor as a "closer", using connections 1 and 3 of the switch, the program branches to the "1" branch if the switch is depressed, and otherwise to the "0" branch.

If you connect a mini-sensor as an "opener", using connections 1 and 2 of the switch, the program branches to the "1" branch if the switch is depressed, and otherwise to the "0" branch.

The next element in our garage door control system is a Motor element. Insert it into the program as you did with the previous two elements, this time under the Branch element. It is best to place the element in such a way that that it is automatically connected to the element above.



The Motor element allows you to switch on or off either a motor, or a lamp or an electromagnet. Again, you open the Properties window for the Motor element by right-clicking on the element.



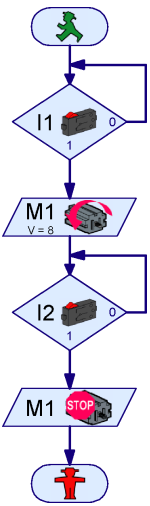
- You can choose which of the Interface's outputs to control by means of buttons **M1** to **M4**.
- Under **Image** you can choose an image to represent the fischertechnik component connected to the output.
- We will deal with the selection **Interface / Extension** when we get to Chapter 77 *Controlling several Interfaces* on page 70.
- Under Action you can select how the output is to be affected. You can start a motor with direction left (counterclockwise) or right (clockwise) or stop it. You can switch a lamp on or off.
- Under **Speed/Intensity** you can set the speed at which a motor is to operate, or how brightly a lamp should glow. Possible values are 1 to 8.

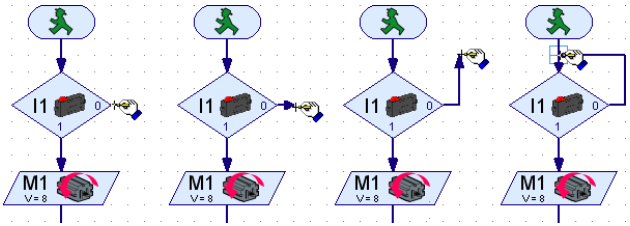
For our flow chart we need the command ***Motor M1 left with speed 8***.

3.4 Linking program elements

Now that you know how to insert elements into a control program, we can get on with the job of completing our control program. Think back to the functional description of the garage door control system: is there still something missing? Right: we may have turned the motor on by pushing the button, but once the door is opened, the motor must be automatically switched off again! In practice, this is done with the so-called "end switch". This is a sensor fitted to the garage door in such a way that it is operated the moment the motor has fully opened the door. As in the case of switching on the motor, this signal can be used to switch it off again. To query the end switch we can use the Branch element again.

So insert another Branch element into the program, one which will check the end switch on input I2. Don't forget to left-click on the element and to set the input to I2. As soon as the garage door is open and the end switch has been pressed, the motor should stop again. This will be achieved using a Motor element. Start with the same element we used to switch on the motor. If you right-click on the element, you can change the function of the element to **Stop motor**. The program is finished off with an End element. Your program should now look almost like the illustration on the right. If you have placed the elements under one another with a separation of one or two grid points, most of the entries and exits will be connected with program flow arrows. But the No (N) exit of the two Branch elements is not yet connected. As long as the switch on input I1 has not been pressed, the program should go back and query the switch again. To draw this line, click with the mouse successively on the places shown in the diagram below.





Hint: It a line should ever not correctly joined to a connection or another line, this will be indicated by a green rectangle at the point of the arrow. In this case you have to create the connection by shifting the line or by deleting it and drawing it again. Otherwise the program flow will not work at this point.

Deleting program flow lines

Deleting lines works exactly like deleting program elements. Simply left-click on the line, so that it gets marked in red. Now click on the delete (**Del**) key on the keyboard to delete the line. You can also select several lines, if you hold down the shift key (that's the key for shifting between upper and lower case) and then left-click on the lines in succession. Apart from this, you can also mark several lines by drawing a frame around them. Now you can delete all the red-marked lines at once by pressing the **Del** key.

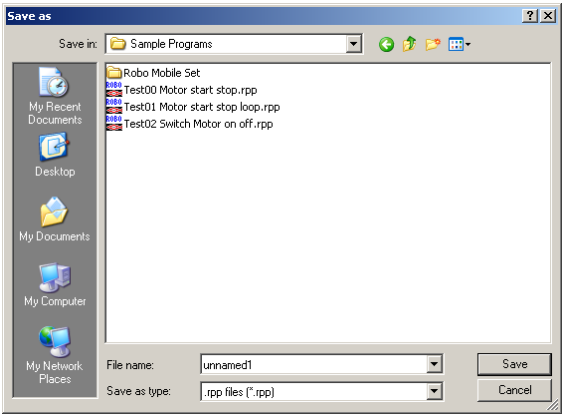
3.5 Testing your first control program

To test our first control program, you should build a little model. To do this, it is enough to connect a switch to I1 and to I2 on the Interface, as well as a motor to M1.

Note: Connecting the Interface to the PC and establishing Interface settings has already been covered in the previous chapter, which you can refer back to for details.

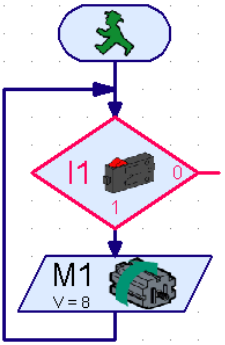
Before testing the program, you should save the program file on the hard drive of your computer. Click on the command **Save as** on the **File** menu. The following dialog window will then appear.

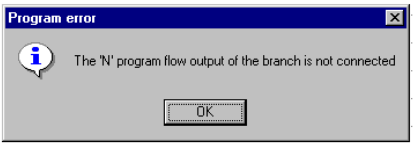
Under “Save in”, choose the directory in which you want to save the file. Under “Filename”, enter a name not yet in use, e.g. GARAGE DOOR and confirm by left-clicking on “Save”.



Start

To test the program, push the start button (shown left) in the toolbar. First, ROBO Pro will test whether all the program elements are properly connected. Should an element not be correctly connected or something else not be in order, it is marked in red, and an error message is displayed describing what is not right. If, for example, you have forgotten to connect the No (N) exit of a program branch, it will look like this:

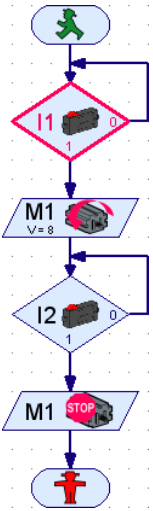




If you have received an error message, you must first of all correct the reported error. If you do not, the program will not be started.

Note: You will find a full explanation of this mode of operation and of “Download Operation” in Section 3.7, on page 24.

The first Branch element will be marked in red. This shows that the program is waiting at this element for an event, namely the pressing of the button on I1, which is supposed to open the garage door. As long as the switch on input I1 has not been pressed, the program takes the No (N) alternative of the branch and goes from there back to the beginning of the branch again. Now press the switch connected to input I1 of the Interface. This fulfils the condition for proceeding, and the motor is switched on. In the next step, the program waits for the end switch on input I2 to be pressed. As soon as you operate the end switch on I2, the program branches to the second Motor element and switches the motor off again. Finally the program arrives at the program end. A message will appear saying that the program has been terminated.

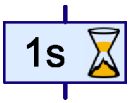


Did everything work? Congratulations! That means you've created and tested your first control program. If it doesn't work properly—don't give up, just check through everything carefully again; there must be a mistake hidden in there somewhere. Every programmer makes mistakes, and making mistakes is the best way to learn. So keep your chin up!

3.6 Other program elements

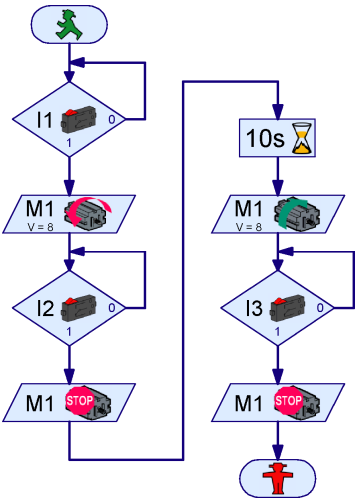
If you have tried your first control program on a real model garage door, the door will now be open. How can we close it again? Of course we can start the motor again by pushing a button! But we want to try another solution, and learn about a new program element in the process. To do this, you start by saving the program under a new name (we will need the current flow chart again later). Use the menu item **Save as ...** in the **File** menu to do this, entering an as yet unused filename.

3.6.1 Time delay

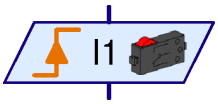


Before we can extend the flow chart, you have to delete the connection between "switch off motor" and "Program end" and shift the End element down. Now you can insert the new program elements between these two elements. The garage door is to be closed automatically after a period of 10 seconds. To do this you can use the **Time delay** program element illustrated right. Within a broad range, you can set the waiting time as you wish, as usual by right-clicking on the element. Enter the desired time delay of 10 seconds. To close the garage door, the motor must of course go the other way, that is, to the right clockwise). The motor is turned off by another end switch on I3.

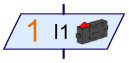
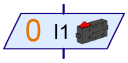
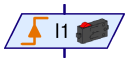
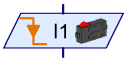
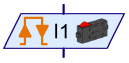
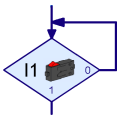
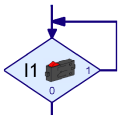
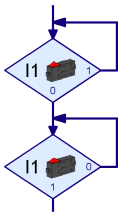
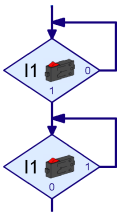
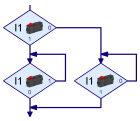
The finished flow chart should look roughly as presented on the right. For the sake of presentation, the new program elements have been moved to the right. Once there are no more mistakes in the flow chart, you can test the extended garage door control system as usual with the **Start** button. The motor is switched on by operating the switch on I1, and switched off again by operating I2. This is how the garage door is opened. Now the Time delay program element has a red border for 10 seconds, that is the delay time we set. Then the motor is switched on to turning the other direction until the switch on I3 is operated. You should also try changing the delay time.



3.6.2 Wait for input

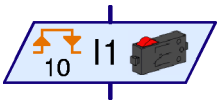


Alongside the Time delay element there are another two elements that wait for something before allowing the program to proceed. The **Wait for Input element**, depicted left, waits until one of the Interface's inputs is in a particular state of has changed in a particular way. There are 5 variants of this element.

Symbol					
Wait for	Input=1 (closed)	Input=0 (open)	Change 0-1 (open to closed)	Change 1-0 (closed to open)	Any change (1-0 or 0-1)
Same function using Branch alone					

A combination of Branch elements could be used instead, but the **Wait for Input** element makes things simpler and easier to understand.

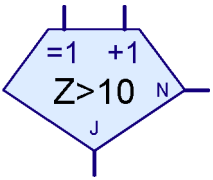
3.6.3 Pulse counter



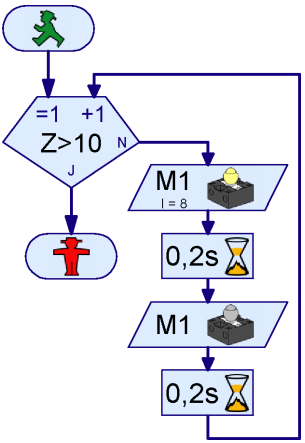
Many fischertechnik model robots also use pulse wheels. These gear wheels operate a switch four times for every revolution. With these pulse wheels you can turn a motor on for a precisely defined number of revolutions rather than for a given time. To do this, you need to count the number of pulses at an input of the Interface. For this purpose there is the ***Pulse counter element***, depicted left, which waits for a user-definable number of pulses. In the case of this element, too, you can set whether any alterations or only 0-1 or only 1-0

changes are regarded as pulses. With pulse wheels, one normally waits for changes in either direction, so that a resolution of 8 steps per revolution is obtained with 4 gear teeth.

3.6.4 Counter loop



With the Counter Loop element you can very easily have a specific part of the program executed several times. The program illustrated, for example, turns a lamp on M1 on and off again 10 times. The Counter Loop element has a built-in counter. If the counter loop is entered via the **=1** entry, the counter is set to 1. If the counter loop is entered via the **+1** entry, 1 is added to the counter. According to whether the counter is greater than a value you have prescribed, the counter loop branches to the Yes (Y) or No (N) exit. So the Yes exit is used when the loop has been traversed as many times as you specified in the counter value. If further passes through the loop are needed, on the other hand, the counter loop branches to the No exit. As in the case of the Branch element, you can also swap the Yes and No exits through the property window.



3.7 Online and download operation—what's the difference?


Start

So far we have tested our control programs in what is called **online operation**. In this way you were able to follow the progress of the program on the screen, because the currently active element was marked in red on the screen. You use online operation to understand programs or to look for errors in programs.


Pause

In online operation you can also stop the program and continue it again by pressing the **Pause** button. This is very practical if you want to investigate something about your model without stopping the program altogether. Also, if you are trying to understand the way a program runs, the Pause function can be very helpful.


Step

With the **Step** button, you can execute the program in individual steps, element by element. Every time you press the Step button, the program goes to the next program element. If you execute a **Time Delay** or **Wait for** element, it can of course take a while for the program to get to the next element.


Download

For your ROBO TX Controller you can also use **download operation** instead of online operation. In online operation programs are executed by your computer. In this mode, it sends control commands such as "switch on motor" to the Interface. For this, the Interface needs to be connected to the computer for as long as the program is running. On the other hand, in download operation the program is executed by the Interface itself. Your computer stores the program in the ROBO TX Controller. As soon as this has been done, the connection between the computer and the Interface can be broken. Now the Interface can execute the control program independently of the computer. Download operation is important for example in programming mobile robots, for which a connecting cable between PC and robot would be very cumbersome. Even so, control programs should initially be tested in online operation, as possible errors are more easily found here. Once fully

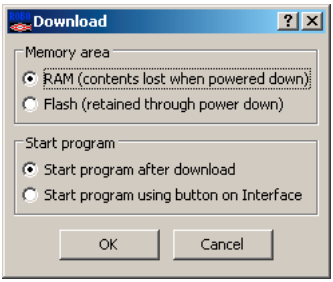
tested, the program can be downloaded onto the ROBO TX Controller. The problematic USB cable can be replaced by a Bluetooth connection. In that way the model has unrestricted mobility even in online operation (see manual ROBO TX Controller).

But online operation also has advantages compared with download operation. In comparison with the Interface, a computer has much more working memory, and can calculate much faster. This is an advantage with large programs. Also, during online operation a ROBO TX Controller and a ROBO Interface can be controlled simultaneously from a program.

An overview of the two modes of operation

Mode	Advantage	Disadvantage
Online	Program execution can be followed on screen Execution, even of large programs, is very fast Simultaneous control of a ROBO TX Controller and a ROBO Interface possible The earlier Intelligent Interface is supported Panels can be used The program can be stopped and continued	Computer and Interface must remain connected
Download	Computer and Interface can be separated after download	The earlier Intelligent Interface is not supported Program execution cannot be followed on screen

Using download mode



If you have the ROBO TX Controller or the ROBO Interface, you can transfer the garage door control system to the interface by means of the **Download** button. First the dialog window on the left is displayed. The ROBO Interface has several program storage areas, a ***RAM*** (Random Access Memory) area and two ***Flash*** memory areas. A program in RAM is lost as soon as you disconnect the Interface from the power supply or the battery pack is discharged. A program stored in Flash memory, on the other hand, will remain stored in the Interface, even without power, for years. Of course you can nevertheless overwrite programs in Flash memory at any time. Download to RAM, however, is distinctly faster, and is therefore recommended for testing purposes.

You can store multiple programs, for example multiple behavior modes for a mobile robot, in the Flash memory. You can select, start and stop the multiple programs by using the display and the selection keys of the ROBO TX Controller. If the ***Start program after download*** option is active, the program is started immediately after download. To stop the program you press the left selection key on the TX Controller.

For mobile robots, the option **Start program with key on Interface** makes more sense. This is because, if you don't have a Bluetooth interface, you still have to unplug the USB cable before your program sets the robot in motion. In this case, you start the downloaded program by using the left selection key of the TX Controller.

By using the function Autostart of the ROBO TX Controller, a program is started automatically as soon as the Interface is supplied with power. In this way, you can for example you can supply the Interface with power via a mains adapter with a time switch, and start the program every day at the same time. Then you don't have to either leave the Interface permanently switched on or start the program with the selection key every time you switch it on.

Note:

You can also find a comprehensive description of the functions of the ROBO TX Controller in the accompanying operating manual.

3.8 Tips and Tricks

Altering connection lines

If you shift an element, ROBO Pro will try to adjust the connecting lines in a reasonable way. Should you not like an adjusted line, you can easily change the connecting lines by left-clicking on the line and moving it while holding the key down. According to where the mouse is placed on the line, a corner or an edge of the line is moved. This is displayed by different mouse-cursors:

If the mouse is positioned over a vertical connection line, you can move the whole vertical line while holding down the left mouse key.

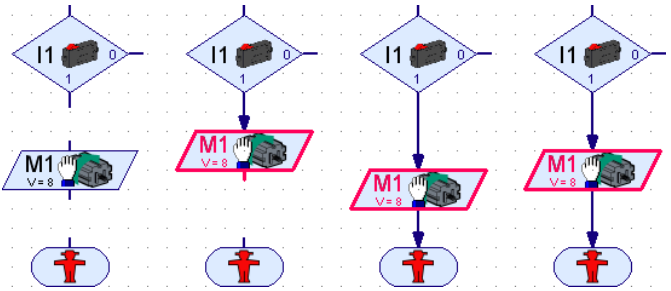
If the mouse is positioned over a horizontal connection line, you can move the whole horizontal line while holding down the left mouse key.

If the mouse is positioned over an oblique connection line, a new point is inserted into the connecting line when you left-click. You have to hold the left mouse key down, not releasing it until the mouse is positioned where the new point is to be placed.

If the mouse is positioned near an end point or a corner of a connecting line, you can move this point while holding down the left mouse key. You can only move a connected line endpoint to another suitable program element connection. In this case the endpoint of the connecting line will be linked to this connecting line. Otherwise, the point will not be moved.

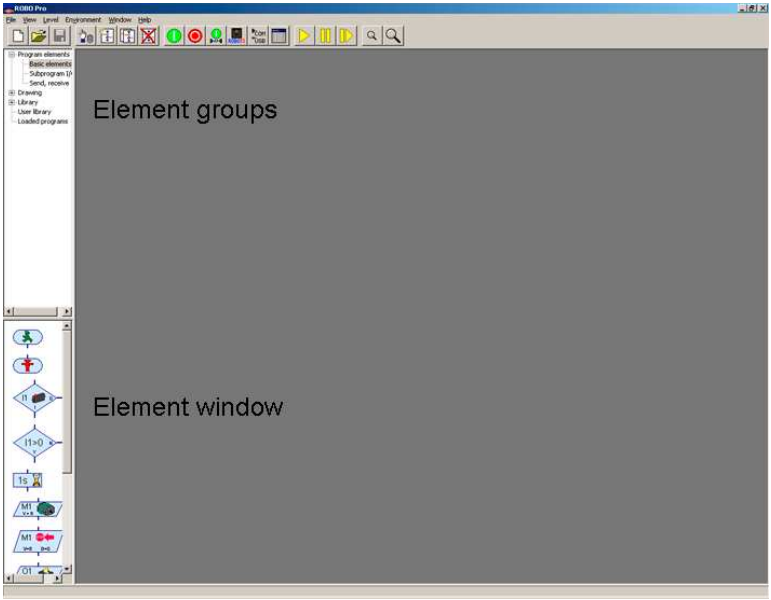
A different approach to connecting lines

Connecting lines can also be created by moving program elements. If you move a program element so that its entry is one or two grid points below the exit of another, a connecting line between the elements is created. This also applies to an exit that is moved over an entry. After that, you can move the program element to its final position or draw further links for the remaining entries and exits.



4 Level 2: Working with subprograms

Once you have successfully created and tested your first control program, you are ready for ROBO Pro Level 2. Now choose the entry **Level 2: subprograms** in the **Level** menu. You are sure to notice the difference straight away: The element window has disappeared, and in its place you now have two stacked windows on the left-hand edge.



But don't worry! The element window is still there, only it's now empty. In Level 2 there are more program elements, so that you would lose track of them if they were all packed into one window. For that reason, from Level 2 onwards, all the elements are classified into element groups. The elements are organized into groups in a similar way to how files on your computer's hard disk are organized into folders. If you select a group in the upper window on the left-hand side, all the elements in this group appear in the lower window. You will find the elements from Level 1 in the group **Program elements / basic elements.** Since the element window is now only half as big, you have to use the scroll bar on the left of the element window to display the lower elements.

So, now let's get down to the real topic: subprograms! Of course the flow charts we have designed so far have not been on such a large scale that we could not take them all in at once, but surely you can imagine that this could very easily happen in the case of larger projects with more complex flow charts. Suddenly your worksheet is full of components, there are connecting lines everywhere and you have to constantly scroll back and forth on the screen. "Now where was this or that exit?" In short—minor chaos threatens! What to do? Is there no way to bring some order into this chaos? Yes there is—it's called ***subprograms***!

4.1 Your first subprogram


New


SP New

A subprogram is very similar to the programs you're already familiar with. To investigate them more closely, first you have to create a new program and a new, empty, subprogram within this program. To do this, press Program **New** and then the **SP New** button in the toolbar. A window will appear, into which you can enter the name of the subprogram and a description of it.

The name should not be too long (ca. 8-10 letters), as otherwise the subprogram symbol will be very large. Of course, you can later modify any entries you make here.

As soon as you close the **New subprogram** window with **OK**, the new subprogram will appear in the subprogram bar.

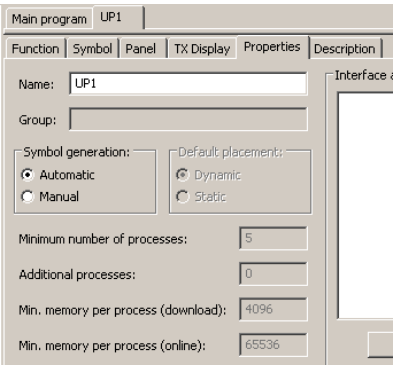
You can switch between the main program and the subprogram at any time by clicking on the program name in the subprogram bar. As both programs are still empty, however, you won't see any difference yet.

We now want to divide the garage door control system from the previous chapter (see Section 3.6 *Other program elements* on page 22) into subprograms. The program consists of four functional units:

- Wait until button I1 is pressed
- Open door
- Wait ten seconds
- Close door

Now we want to separate opening and closing off into two subprograms. Either subprogram can then be called from the main program with a single symbol. The waiting for sensor I1 and the 10-second time delay remain in the main program, as each of them consists of only a single element anyway. You have just established a new program with a subprogram named **Subprogram 1**. However, **Open** and **Shut** would be better names for these two subprograms. You can rename the already created subprogram by first selecting Subprogram 1 via the subprogram bar, if it is not already selected.

Then switch via the function bar to the Properties window for the subprogram by clicking on **Properties**. Here you can change the name of **SP 1** to **Open**. Most of the other fields can only be altered in the Advanced or even Expert levels. The item **Symbol creation** will be explained later on.



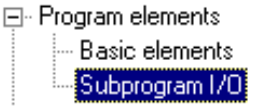
If you click on Description on the function bar, you can change the previously entered description, although “My first subprogram” remains an accurate description.

In the function bar, click on **Function** now, so that you will be able to program the function of the subprogram. Now you will see the program window again, in which you inserted program elements in the previous chapter for your first ROBO Pro program. Make sure that you have selected the subprogram **Open** in the subprogram bar.

Are you ready to write your first subprogram? Well let’s go! But just how does a subprogram start? Good question! You have always begun a main program with the start element. A subprogram begins with a similar element the Subprogram Entry. The element has this name because it is through this element that program control passes from the main program into the subprogram. You can’t use a Start element here, because of course no new process is being started.

	Start element	Starts a new, independent process.
	Subprogram entry	Here program control is handed over from the main program to the subprogram

You will find the Subprogram entry in the element group window under **Subprogram I/O**. Now place the Subprogram entry near the top of the program window for the **Open** subprogram. You may also give a Subprogram Entry element a different name than **Entry**, but this will only be necessary if at some later time you write a subprogram with multiple entries.

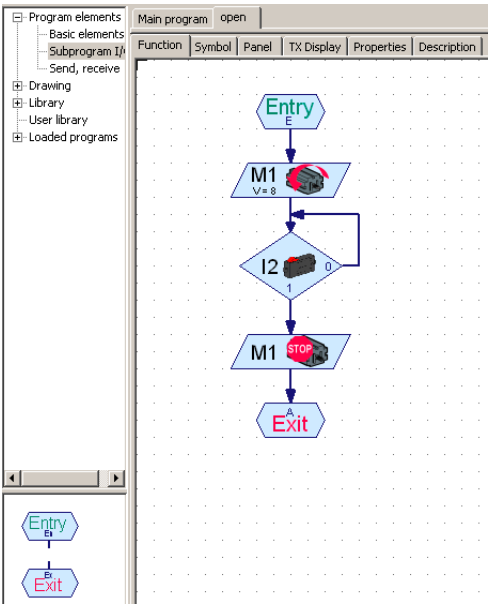


The subprogram now runs identically to the part of the main program which does the opening. You switch on the motor M1 in the left-rotating (anticlockwise) direction, wait until the sensor on input I2 is closed and then switch the motor off again.

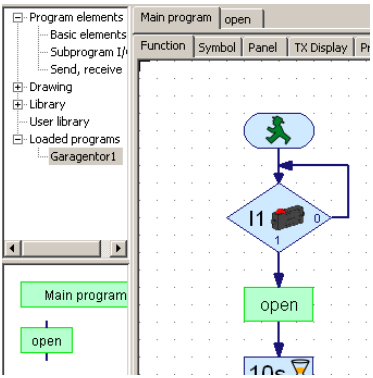
To close off the program you use a Subprogram Exit. The difference between the Subprogram Exit and the Stop element is the same as between the Subprogram Entry and the process Start.

	Stop element	Stops program execution of an independent process
	Subprogram Exit	Here program control is handed back from the subprogram to the main program

The completed subprogram should now look something like this:



Make sure you have actually entered the subprogram under **Open**, and not under **Main program**. Now switch in the subprogram bar from **Open** back to **Main program**. Now you will see the main program program window, which, as before, will be empty. As usual, insert a Start element (not a Subprogram entry!) into the main program. Querying the switch on **I1**, which is supposed to open the garage door, you will also do as before in the main program.



Now you can insert your new subprogram, like an ordinary program element, into your main program (or into another subprogram). You will find it in the element group window under **Loaded programs** and the filename of your program. If you have not yet saved your file, it has the name **unnamed1**. If you have loaded other program files, you can also select subprograms belonging to other files in the selection window. This way, it is very easy to use subprograms from another file.

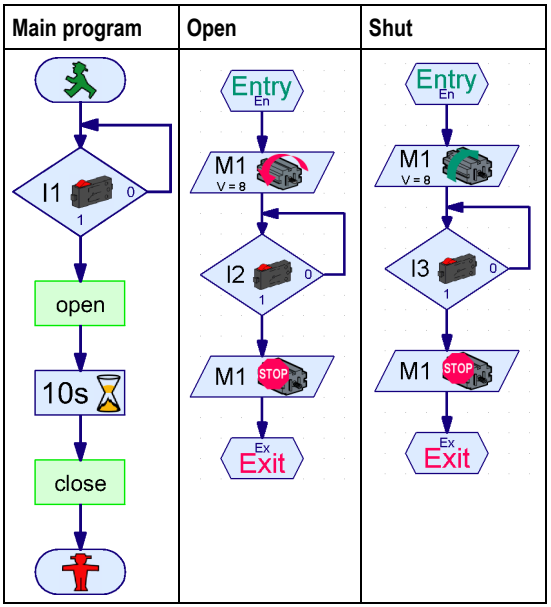
In the element group **Loaded programs / unnamed1** you will find two green subprogram symbols. The first, with the name **Main program**, is the symbol for the main program. This is used rather infrequently as a subprogram, that even that is possible, for example if you are controlling a whole machine park, and you have previously developed the control systems for the individual machines as main programs. The second symbol, with the name **Open**, is the symbol for your new subprogram. **Open** is the name you entered under Properties. Now insert the subprogram symbol, in the same way as you're used to doing it with ordinary program elements, into your main program. It's as easy as that!


SP New

You can close your main program right now with a stop element and try it out, if you like. The door will be able to be opened by pressing the button on I1, but we haven't programmed the closing part yet. To do that, you write another subprogram. Press the **SP New** button on the toolbar and enter the name **Shut** in the **New subprogram** window. You are not obliged to enter a description, but it wouldn't hurt, so you won't forget later what the subprogram is meant to do.

Now enter the program for shutting the garage door in the program window for the subprogram **Shut**. Once again, you start with the Subprogram entry. First the motor **M1** should turn to the right (clockwise). As soon as the end switch on **I3** is closed, the motor **M1** should stop. Once again the subprogram is closed off with a Subprogram exit.

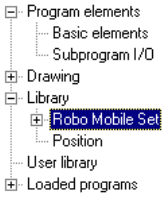
Now use the subprogram bar to switch back to the main program. If you previously closed off the main program with a Stop element so as to try it out, you must delete the Stop element again. After being opened, the garage door should remain open for 10 seconds before being closed again. After a 10-second Time Delay, you insert the **Shut** subprogram symbol from the element group **Loaded programs / unnamed1**. The main program and the two subprograms should look something like this:



The program starts at the Start element in the **main program**. Then it waits until sensor **I1** is depressed. Incidentally, you could also do this using the ***Wait for Input*** element (see Section 8.1.9 *Wait for input* on page 80). After the switch I1 has been pressed, the main program calls the subprogram **Open**. This switches program control to the Subprogram Entry for the subprogram **Open**. The subprogram **Open** opens the garage door and then reaches its Subprogram exit. At this point program control returns to the main program. After the subprogram finishes the main program waits for 10 seconds. Then program execution switches to the subprogram **Shut**, which shuts the garage door again. After control returns from the subprogram **Shut**, the main program comes to a Stop element, which terminates the program.

4.2 The subprogram library

It is very easy to copy subprograms from one file to another: you load both files and insert a subprogram from one file into another using the element group **Loaded programs**. For frequently used subprograms, however, the process is even simpler, through use of the **Library**. ROBO Pro contains a library of ready-made subprograms that you can easily re-use. As well as that, you can create your own library, in which you can store your frequently used subprograms.

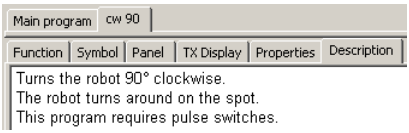


4.2.1 Using the Library

The **Library** is initially divided into two main groups. In the **Construction kit** group you will find subprograms you can use for models from specific construction kits. In the **General** group you will find subprograms you can use for all possible models. But most of these subprograms in the **General** group require techniques from Level 3, which are not explained until the next chapter.

Each computing Construction Kit, for example the ROBO Mobile Set, has its own subgroup in the Construction Kits group. This is sometimes further subdivided according to the models you will find in the construction manual for the Construction Kit. When you select the Construction Kit or one of the models, the ready-made subprograms for this model are displayed in the element window.

If you point to one of the subprogram symbols with the mouse, a short description is displayed. If you insert a subprogram into your program, you can display a detailed description by selecting the subprogram in the subprogram bar and then clicking on **Description** in the function bar.



Caution: If you insert a program from the Library, in some cases further subprograms that are used by this subprogram will also be inserted. You can remove all these subprograms again by selecting the Undo function on the Edit menu.

4.2.2 Using your own library

After you've been working away with ROBO Pro for a while, you are sure to have some subprograms that you use more frequently than others. To avoid having to look for and load the relevant file every time, you can also set up your own subprogram library, which functions in exactly the same way as the pre-defined library. Your own library consists of one or more ROBO Pro files all

stored in one folder. Each file in this folder will be represented by its own group in the group selection display.

You can specify which folder you'd like to store your own library in the **File** menu under **Own library directory**. The default directory for your own library is C:\Programs\ROBOPro\Own Library. If you have your own user directory on your computer, it is a good idea to create your own folder there and use this to store your library.

Tip: Initially you can specify, under Own library directory, the folder in which you also store your ROBO Pro programs. That way you will have rapid access to all subprograms in all files in your working directory.

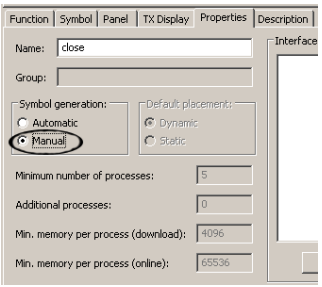
Organizing your own library

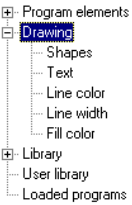
ROBO Pro has no special functions to alter a library. But the procedure is quite simple. If you would like to add subprograms to a library group or remove them from one, you must first load the corresponding file. You will find this file in the directory which you have established as your **Own library directory**. Now you can, for example, load a second file and drag a subprogram from this from the **Loaded programs** group into the main program of the library. In a library, the main program is not a real program, but rather just a collection of all the subprograms in the library. In the case of libraries, the main program itself is not displayed in the element window. Of course you can also delete subprograms from a library or modify subprograms there.

If you have modified a library file and saved it, then you must select the menu item **Update own library** in the **File** menu. This will update the file list in the group window.

4.3 Editing subprogram symbols

As you saw in the previous section, ROBO Pro automatically generates a subprogram symbol for your subprograms. But you can also draw your own symbols, which give a better idea of what your subprograms do. To do this, you must switch from automatic to manual symbols in the subprogram's Properties window. Next you can switch from **Properties** to **Symbol** in the function bar and edit the subprogram symbol there. You will find drawing functions in the element group window under **Draw**.

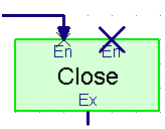




Under **Draw / Shapes** you will find all the usual basic graphic elements such as rectangle, circle, ellipse, polygons, etc. Under **Draw / Text** you will find text objects in various font sizes. In the other groups under Draw you will find functions to alter the color and similar properties of selected elements. Precise details of how to use the drawing functions are given in Chapter 10 *Drawing functions* on page 121. Also observe the functions in the main menu under **Draw**.

You can also move the connections of the subprogram, but you can't delete the connections or add new ones. In the subprogram symbol there is always one connection for each subprogram

entry or exit. The connection elements are generated automatically, even if you have switched to manual symbol generation.



As soon as you leave the symbol-editing window, all calls to the subprogram in the main program or in other subprograms will be modified accordingly. Please take note that, if you have moved the connections of a subprogram, this can cause a little confusion with subprogram calls, if the connections were already connected. The endpoints of the connecting lines may in some circumstances no longer occur at the right connection, which will be shown by a cross at the endpoint of the line and at the connection (see diagram). As a rule, it is generally sufficient to left-click anywhere on the connecting line. The line will then be automatically re-aligned. But it can happen in the case of subprograms with a lot of links that you will have to edit the line further.

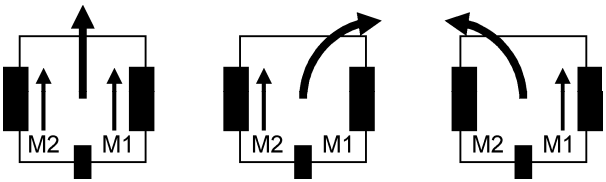
4.4 Tango

So far, you only became acquainted with rather simple programs and maybe you are eagerly waiting for new program elements and possibilities. But before we deal with variables and still more difficult things in the next chapter, let us first see everything what can be done with the program elements in the level 2. How would it be, for example, if would provide tangoing to your mobile robot? For the “Nerdse" among you: Tango is danced to music in the 2/4 measure. The basic step includes 8 steps in 3 measures. For the gentlemen, the step sequence is as follows:

- One slow step forward with the left foot (1/4 measure)
- One slow step forward with the right foot (1/4 measure)
- Now comes the continuous 4/8 measure „swing step“. Thereby you move the feet a little to not at all, but only displace your weight. At first, you displace your weight by 1/8 measure on the left back foot, then by 1/8 on the right fore foot and then again by 1/8 on the left back foot. For finishing the swing step, you take a break of 1/8 measure.
- Three quick steps follow: First, make a small step backwrds with the right foot, so that it lays again near the left foot. Then you make a step aside with the left foot and, in the end, you place again the right foot near the left foot. These three steps also last 1/8 measure each and they will be finished again with a 1/8 measure break.

For the lady, the step sequence is symmetrical, that means left and right, as well as forward and backward are inverted. The whole repeats till the music comes to the end, you hit on the borders of the room or it becomes boring to you. In both the last cases, you should ask a dancing master for advice.

But now again to robotics. Maybe you have a Fischer technique building set for mobile robots. The robots in this set mostly have two driving wheels, with an independent motor each. Guiding is made by these robots in the same way as for track vehicles. If you turn both driving motors in the same direction, the robot moves straightforward. If a motor is at rest, the robot runs a curve.



Naturally, with these robots you can also move backward, straightforward and about the curve. If both the driving engines turn in opposite direction, the robot turns in place. Let us now try to translate the tango step sequence in wheel turns. One 1/4 measure should thereby last one wheel turn. We get then:

- Left wheel 1 turn forward (usually motor M2 left).
- Right wheel 1 turn forward (usually motor M1 left).

Now comes the „swing step“. But naturally, our robot can not move the body without moving the "feet". Also, the side step in the 3-rd measure is quite difficult for a robot. Therefore we make a light turn left in the 2-nd measure and move then in the 3-rd measure a small portion straightforward, for simulating the side step. For the 2-nd measure, it results:

- Left wheel ½ turn back (usually motor M2 right).
- Right ½ turn forward.
- Left wheel ½ turn backward.

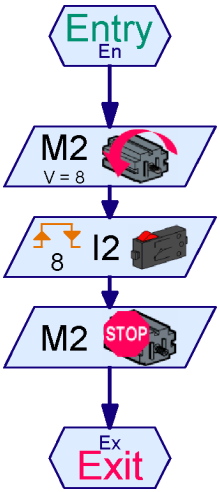
Both on „left backward“ and on „right forward“ the robot turns left. In the 3-rd measure, we make now the following:

- Right wheel ½ turn backward.
- Straightforward ½ turn forward.
- Right wheel backward and left wheel forward for ½ turn.

Consequently, first we turn again the robot quite a little to the right, then we move straightforward (in foreward left direction), for simulating the side step to the left, then we turn again the robot straight.

Now let's try to perform this sequence of steps in ROBOPro. The form of execution will differ, depending on whether you are using a TX controller with encoder motors or a model with pulse switches. The two cases are described separately below: 4.4.1 *Motor control with pulse switches* on page 37 and 4.4.2 *Motor control with encoder motors* on page 40.

4.4.1 Motor control with pulse switches



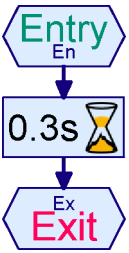
At the best, you begin with a sub program for the single steps. A subprogram for the first step "Left wheel 1 turn“ is shown on the right. Usually, the driving motor for the left wheel is connected to the M2 interface output and the appropriate pulse switch to the interface input I2, whereupon counter-clockwise is forward.

For the first step, you switch the motor M2 counter-clockwise (full speed) and you wait then 8 pulses at the I2 input. 8 semi pulses means that you count both 0⇒1 and 1⇒0 flanks. You can choose the element in the property window of the pulse counter. In many models, 8 semi pulses correspond to one wheel turn. But this can also differ depending on transmission and arrangement of the pulse switches, for example 16 semi pulses per turn.

As soon as the 8 pulses are entered, you disconnect again the M2 motor and the sub program is finished. You may call this sub program, for example „Links 1/4.“

For the further steps, you still need the following sub programs:

- Right 1/4 (As Left 1/4 with M1 and I1 instead of M2 and I2)
- Left 1/8R (As Left 1/4, but 4 instead of 8 semi pulses and backward, consequently the motor clockwise)
- Right 1/8 (As Right 1/4, but 4 instead of 8 semi pulses).
- Right 1/8R (As Rights 1/8, but backword, while motor clockwise)



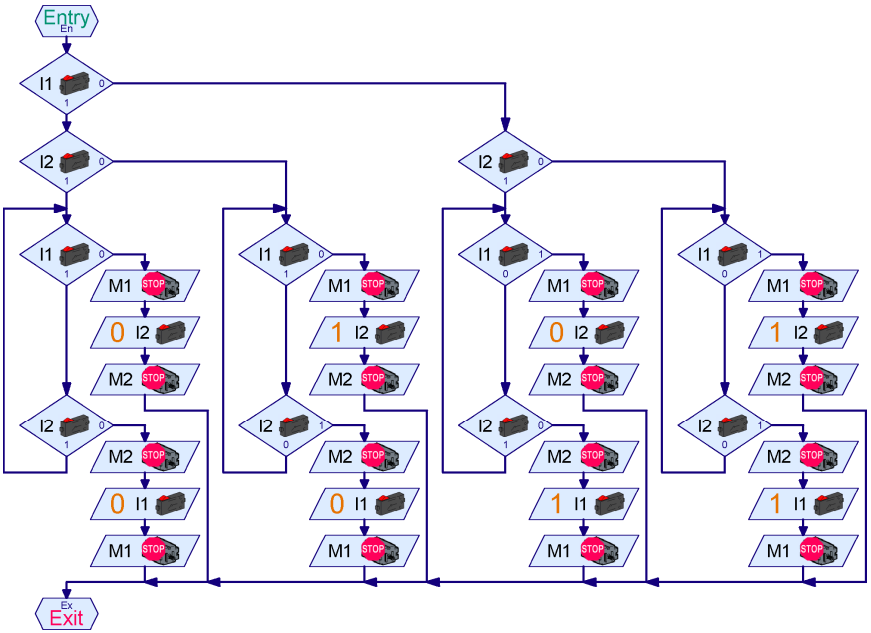
Naturally, you can not wait 1/8 pauses over pulse counter, bacause no wheel moves in the pauses. Instead of that, we use a delay time. With the standard models in the ROBO Mobile Set, 4 semi pulses correspond to about 0,3 seconds. But depending on translation and motor, it may be also otherwise with your model. Also provide a sub program for the 1/8 pause. The sub program contains, apart from the sub program input and output, only one single program element, but you need the pause twice. If you apply a sub program for it, you can easier change the pause time.

Now maybe you will argue that we should also use a delay time for the steps and no counter. Then there would not be the problem to adapt the pause time and the step time. But the disadvantage would be that the right motor and the left motor never turn quite equally fast and therefore the robot would not dance reproducible forms. When you use the pulse switch, on the contrary, you make sure that both the wheels move ever exactly the same way, even if the accumulator is discharged or a wheel moves a little more difficult than the other.

Now we still miss the sub programs for the 1/8 straight step and the 1/8 turn on the spot. In fact, it should work exactly as the other steps, only that you switch on two motors instead of one. The only question is how is it done with the puls switches. Simply set two puls counters, one after the other, will not do. Then the program would first wait 4 half-steps of one motor and then 4 half-steps of the other motor. Waiting on only one of the two motors, we would approach the matter, but then there would be problems if both the motors would not turn equally fast. The best solution is to start both the engines and then wait till one of the pulse switches changes. Then you stop immediately the motor the pulse switch of which has changed and wait then till the second pulse switch changes

and you can stop the second motor. In all, it is unfortunately somewhat complicated, because you do not know whether the pulse switches are open or closed at the beginning. Because two pulse switches exist, there are in all four possibilities. But fortunately, there is already a completed sub program from the library for this function. Create a “Straight 1/8“ sub program and insert the „SyncStep“ sub program from the homonymous library in the “ROBO Mobile Set” folder. If you do not know any more how it goes on, please check up in the chapter 4.2 *The subprogram library*, at page 33.

Now for the inquiring people among you, the „SyncStep“ sub program represented below will still be also shortly explained. The people for whom a look at the sub program is sufficient may willingly overleap the following paragraph and the duty. It is entirely all right to use a sub program without understanding how it works -- as long as you understand what it does.

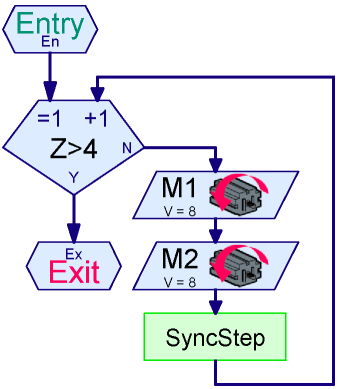


The sub program first asks about the states of both the pulse switches, I1 and I1. Depending on whether I1 and I2 are 0 or 1, the sub program jumps in 4 different program branches. The branch quite on the left is for the case in which I1 and I2 have the 1 value at the beginning. The sub program must then wait, in a logical way, that I1 or I1 should receive the 0 value. This will be done in the loop with the two branching elements. As long as I1 and I2 have the 1 value, the program turns in a circle. But as soon as one of the two inputs becomes 0, the respective motor will be immediately switched off. Then the sub program waits with a „Wait on input“ element until the second input becomes 0 and then switches the second motor off. The loop in which it is being waited on both inputs is necessary, because you do not know which of the two motors turns faster and which of the pulse switches changes therefore faster. The other 3 branches operate exactly so, but they start from another initial state and are therefore waiting the final state respectively opposite to the initial state. For example, in the second branch from left, at the beginning, I1=1 and I2=0, as you can test easily, by pursuing the way via the first two branching elements. Consequently, the second branch waits that the values should become I1=0 and I2=1. If you would like to write the

program yourself, you must watch out very exactly which are the initial values of the pulse switches in each branch, and which you must wait accordingly.

If you have already browsed something over variables in the following chapter, try once to write a sub program with same function with variables. That is easier, because you can save at the beginning the value of the two pulse switches with the = command elements in two variables and you only need one program branch, in which you can compare the current value of the inputs with the values of variables.

So, now back to the tango: The purpose of the SyncStep sub program consisted in writing therein a „1/8 Straight line“ sub program, which goes 4 half-steps straightforward. If you start the motors M1 and M2 and you execute then the SyncStep sub program, the motors are again stopped after a half-step. Consequently, you must make all of it 4 times, and it runs at the best with a loop element.

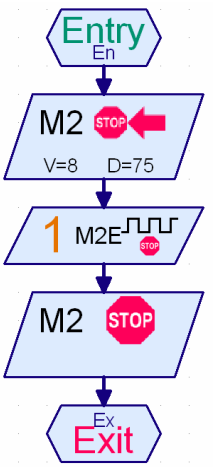


If you watched out very well, you now probably worry that the motors, at the end of the SyncStep sub program are switched off, and then switched again on immediately. With the slower of two engines there is a very short break between switching off and on, which is necessary to adapt the speeds of the two motors to each other. That is however harmless for the motors. In fact, the interface regulates the motor speed also by constant switching off and on. That is called the PWM (pulse width modulation). On the contrary, with the faster motor, switching off and on occurs so fast that the motor takes absolutely no notice of it. However, in the SyncStep sub program you could also give up switching off the second motor and switch off both the motors as soon as the loop is completed. With programming too, different ways often lead to the goal.

Try out whether a robot with the SyncStep sub program really runs better straightforward, than if you simply switch on a certain number of pulses on both the motors.

The last sub program, which we still need, should turn the robot 4 half-steps long to the right on the spot. It is interesting that you can use for that exactly the same “SynchStep” sub program as for the “1/8 Straight line” sub program. „The SyncStep “sub program namely stops the motors only, and the stop command does not depend on the direction of rotation. You start the motor M1 in the loop simply with a direction of rotation right instead of left. The pulse switches are also independent of the direction of rotation. It is all the same whether the motors turn to the left or to the right, the pulse switches always change from 0⇒1 and from 1⇒0. Consequently, in order to create the „Turn 1/8” sub program, you only need to copy the „1/8 Straight line “ sub program and to change the direction of rotation.

4.4.2 Motor control with encoder motors



You should preferably begin with subprograms for the individual steps. A sub-program for the first step "Left wheel 1 turn" is shown on the right. Normally the drive motor for the left wheel is connected to the M2 interface output and the corresponding pulse switch to I1 interface input, with counter-clockwise being forward

For the first step, switch motor M2 clockwise (full speed) and then wait 75 full pulses at input C2. 75 full pulses means that you wait 75 times for the change 0➔1 followed by the change 1➔0. Encoder motors have **advanced motor control** that starts the motor and stops it again after a set number of pulses. To this end, select the **Action Distance** in the properties window and enter 75 pulses as **Distance**.

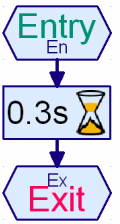
But the element does not have to wait for the motor to reach its target. The program could do other things while the motor is running. However, in this case we just want to wait for the motor to reach its target. To this end, every motor output has its own "target reached" input. For motor M2, this is input **M2E**.

By the way, an encoder motor needs two connections at the TX controller, a motor output M1 to M4 and a counter input C1 to C4. An encoder motor always uses the counter input with the same number as the motor output. This is why the counter numbers cannot be adjusted in the properties window of the advanced motor control.

Once the motor has reached its target, delete the **Action Distance** because the motor control stops the motor on reaching its target. The motor then no longer reacts to normal motor commands such as Left or Right. To do this, use the **advanced motor control** again, but this time with the **Action Stop.** However, this is only necessary if you want to control the motor with the normal motor element. The motor reacts to actions in the **advanced motor control** also without **Action Stop**.

For the further steps, you need the following subprograms:

- Right 1/4 (as left 1/4 with M1 and M1E instead of M2 and M2E)
- Left 1/8R (as left 1/4 but 37 instead of 75 pulses and backwards, i.e. motor clockwise)
- Right 1/8 (as right 1/4 but 37 instead of 75 semi pulses)
- Right 1/8 R (as right 1/8, but backwards, i.e. motor clockwise)

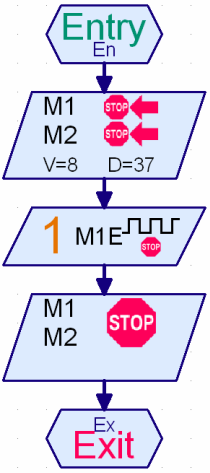


Of course you cannot wait for 1/8 pauses using the pulse counter because no wheel moves in the pauses. Instead, we use a delay time. With the standard models in the ROBO TX Training Lab, 37 pulses correspond to about 0.3 seconds. But this can differ in your model, depending on transmission and motor. So you need to write a subprogram for the 1/8 pause. Apart from the subprogram input and output, the subprogram contains only one single program element, but you need the pause twice. If you create a subprogram for this, it is easier for you to change the pause time.

Now you could argue that we should also use a delay time for the steps instead of the advanced motor control. This would avoid the problem of having to adapt the pause time and the step time. But the drawback would be that the right and left motor never turn at exactly the same speed so

that the robot would therefore not dance in reproducible forms. On the other hand, using the advanced motor control ensures that the two wheels always cover exactly the same distance, even if the battery is flat or one wheel is stiffer than the other.

Now we just need the subprograms for the straight 1/8 step and the 1/8 turn on the spot. The advanced motor control also offers the possibility of controlling two motors at the same time, with the **Synchronous** and **Synchronous Distance** actions. The **Synchronous** action ensures that two encoder motors turn at exactly the same speed. As a result, your robot moves almost straight ahead. However, exactly straight ahead is not possible with encoder motors, because the wheels always have a certain slip. In our case, M1 and M2 should turn at the same speed over a distance of 75 pulses. To do this, use the **Synchronous Distance** action. For motors with synchronous coupling, the target reached signal is not set for both motors until both motors have reached their target. It is therefore sufficient to wait for one of the two target reached signals. Don't forget to stop both motors again at the end!



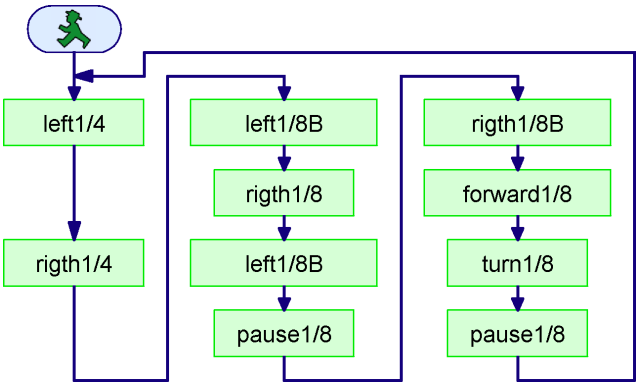
Try out whether a robot really does move straight ahead with more accuracy using the **Synchronous Distance** action compared to controlling both motors separately with the **Distance** action.

The last subprogram that we need is supposed to turn the robot to the right on the stop for 37 pulses. Here again, use the advanced motor control with the **Synchronous Distance** action, with both motors turning in different programs for the subprogram "Turn 1/8": Think about which way M1 and M2 have to turn for the model to turn to the right on the spot.

4.4.3 Tango main program

Now, after you have all sub programs together, you can begin the main program. Now, it is not at all no longer difficult. How the main program could look, you see above on the next page

The shown main program runs the Tango steps endlessly in a loop. Try to use a counting loop for executing the whole tango sequence 5 times in a loop. To that purpose, copy the content of the main program with Processing/Copying and Processing/Inserting into a new sub program and add an Input and Output sub program. Then you can run this sub program 5 times in the loop.



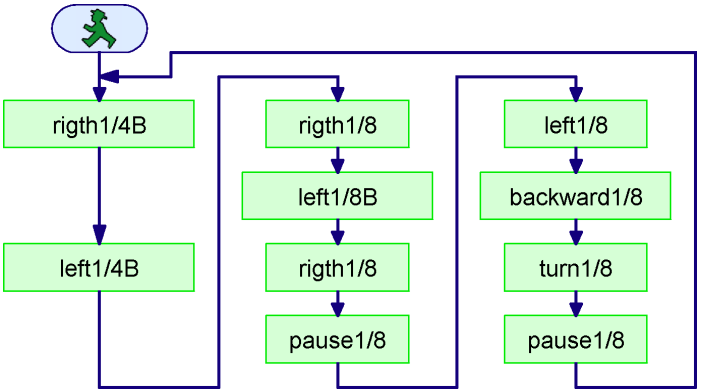
You find the finished tango program in the ROBOPro installation list under:

Sample programs\Manual\Tango Encoder Motor\TangoSolo.rpp

Sample programs\Manual\Tango Pulse Switch\TangoSolo.rpp

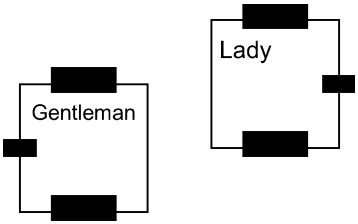
However, if you have a suitable robot, test once your self-written program or the finished program.

Now maybe you think: it is fairly nice, but actually two belong to dancing. Writing a program which executes the lady step suitable for the gentleman step is however not difficult. You must thereto only interchange leftwards and rightwards and forwards and backwards. First load your program for the gentleman step and save it under a new name, for example **TangoSoloLady.rpp**. Now change the sub programs, for example **Left 1/4** to **Right 1/4 B**. For that purpose, you must change M2 to M1 and the direction of rotation from left to right. You can change the name of the sub program by clicking on the properties tab and there enter a new name. The name also changes automatically in the main program where the sub program is called.



The **Turn 1/8** sub program does not change. Can you imagine why? Interchange leftwards and rightwards (M1 and M2 in the sub program) and forwards and backwards (motor left-wards/rightwards) in **Turn 1/8**, and compare the original sub program with the changed sub program.

If you have two mobile robots, now load into one the **TangoSolo.rpp** program and into the other the **TangoSoloLady.rpp** program. If you have only one robot, perhaps you can try it out together with someone which has also a robot. While downloading, you should state that the program is started over the feeler at the interface. Now place both robots against each other as in the drawing below, easily displaced, and start both robots at the same time by briefly pressing the Start feeler (TX Controller) or the PROG feeler (ROBO interface) at both interfaces.

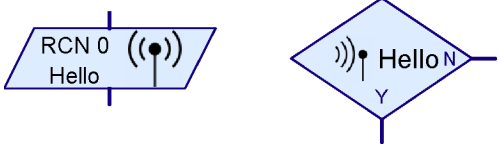


If you have started both robots somewhat at the same time, both will dance tango awhile prettily in time with one another. However, because the motors and accumulators are not exactly alike, the motors do not turn just equally fast and sooner or later the robots come distinctly off tact. How is it done so that that both robots should remain in tact over longer time, you will learn in the next chapter.

4.5 Tango 2: Communication through Bluetooth or RF data link

To remain in rhythm, the two tango robots have to coordinate their movements. The TX-Controller has an integrated Bluetooth radio module. The ROBO RF data link is available for the ROBO interface. The ROBO RF Data Link consists of two radio modules. The interface radio module is built in as card directly into the ROBO interface, the PC radio module in the red housing is connected to the PC over the USB. Up to now, you have probably only used the radio link to manage your mobile robots by radio and therefore without cable connection, online. But Bluetooth and RF data link can do much more: two robots can exchange messages and so communicate with one another.

In the level 2, under **Program items**, there is a **Send, Receive** sub-group, with two elements. The left element in the picture is the transmitter, the right element, the receiver.



The represented transmission element sends the Hello message to the TX-Controller or the ROBO Interface (just "TX-Controller" below) with the radio call number 1, abbreviated to FRN 1. The radio call number is a kind of telephone number, by means of which it is stated to which controller the message is sent. More thereto you learn in the following chapter 4.5.1 *Radio settings for the Robo interface* on page 47 and 4.5.2 *Bluetooth settings for the TX controller* on page 49.

The receiver at right in the picture operates like a program branching: If the **Hello** message was received, the element is branched to the Yes output **Y**, otherwise to the o output **N**.

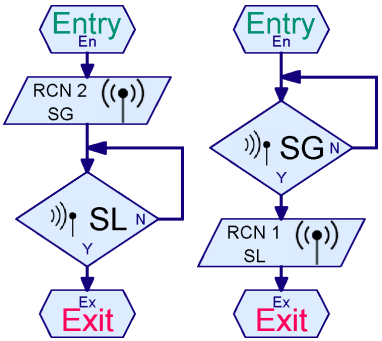
Let us suppose that the transmitter element is called on by a program on the controller with the radio call number (telephone number) 2. It sends then over radio the **Hello** message to the interface with the radio call number 1, because it is indicated in the transmitter as destination. The interface with the radio call number 1 notes that a **Hello** message was received. If the next time a receiver element asks this interface whether a **Hello** message arrived, the answer is Yes, then again No, till another Hello message is received. If on this interface a receiver element next time asks whether a Hello message arrived, the answer is Yes, thereafter again No, until a further Hello message has arrived. The receiver cannot differentiate from which interface the message was sent†. In order to be able to differentiate it, you must send different messages.

Hello = Heli

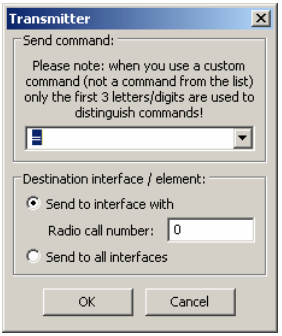
The message is an arbitrary word as in the example **Hello**. However, for technical reasons, only the first three letters or numbers of the message are considered. You can indicate more than three characters, but „Hello“, „Help“ and „helicopter“ all stand for the same message, because all begin with ‚Hel'. Large and lower case and special characters (space character!? , % and the like) are likewise not differentiated. XY! And XY? stand also for the same message. Numbers are however differentiated, so that XY1 and XY2 are different messages.

† Starting from the level 4, there are groups of receivers for this purpose.

Now the synchronisation of the two Tango robots with the transmitter and receiver element is not at all difficult. At the beginning of the step cycle, one robot sends one „Shall we dance“ message and the other robot one „Let us dance“ message and it gets loose. Since the messages should not be so long, we call them SG for „Gentleman synchronisation“ and SL for „Lady synchronisation“. On the right, two sub programs for the synchronisation are represented. The left sub program is executed by the „Gentlemen“ robot with the radio call number 1 and first sends a SG message to the „Lady“ with the radio call number 2. Subsequently, the „Gentleman” waits for a SL message from the „Lady“.



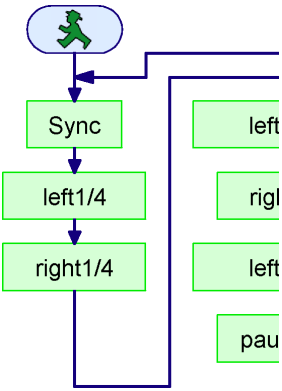
On the right, the characteristic window of the transmitter element is represented. Under the Send command, you can select an command (a message) from the list or enter your own command. Under the destination interface, you can select whether the command should be sent to an interface with a certain radio call number or to all interfaces.



On the left, you see the characteristic window of the receiver element. As with the transmitter, you can select an command (a message). Then you must further select whether the receiver only reacts to commands which were sent directly to the interface, consequently with a certain radio call number, or to commands which were sent to all interfaces. You can also select both. At last, you can still interchange the Yes and No connections, as with any branching element.

So far we have spoken of „Messages“. In the characteristic windows of the transmitter and receiver and later on in the level 3, however the term „command“ will be used. From the view point of the Data transmission, that is the same. Whether a message is an command, depends on the interpretation, and not on the kind of the transmission. In the level 3, you will have very much to do with messages, which are, for example, commands for controlling a motor or a variable. Therefore, in ROBOPro, the term „command “ is generally used for messages.

Insert now into any of the the programs **TangoSolo.rpp** and **TangoSoloLady.rpp** one of the synchronisation sub programs illustrated on the previous side. Call the sub programs **Sync**. You can naturally also write the sub programs in such a way that the „Lady“ invites the „Gentleman“ to dancing. You call the sub program as in the picture on the right in the main program any time at the beginning of the step cycle. You find the finished programs in the installation list of ROBOPro under



Sample programs\Manual\Tango Encoder Motor
Sample programs\Manual\Tango Pulse Switch

TangoSyncGentl.rpp
TangoSyncLady.rpp

Load the two programs, each into a robot, and start the programs. With the ROBO interface and the data link, you'll see that the robots only dance if the program on the "Lady" is started first. That is because the „Gentleman” at first sends a **SG** message, and and the „Lady “waits for this message. If the “Gentleman” is started first, the message goes empty and the „Lady “ waits and waits and waits…. If, on the contrary, the „Lady” is started first, she already waits for the **SG** message when the „Gentleman“ is started. That is naturally somewhat unpractical, in particular when you forgot which robot is the „Gentleman“ and which the „Lady”.

ROBO-IF

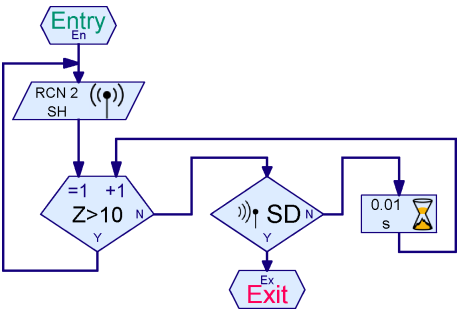
This problem does not arise with the TX-Controller because the gentleman does not send the message until the Bluetooth radio link to the lady has been set up.

TX-Controller

Can't we do something similar for the ROBO Interface?

ROBO-IF

Quite simple: The „Gentleman“ must repeat his invitation for dancing until the “Lady” answers. Any time when the „Gentleman” has sent the message, he waits awhile whether an answer comes from the „Lady“. In the example on the right, the „Gentleman” sends the SG message and then waits in a loop 10 times 0.01 seconds, thus altogether to 1/10 second whether the „Lady“ answers. If the „Lady” does answer within this time, the „Gentleman“ sends again an SG.



Now you probably ask yourself why the gentleman can not simply send a **SG** message in a loop and immediately check whether he received an **SL** message. That is because the transmission of the message from the „Gentleman” to the “Lady“ lasts 1/100 to 2/100 seconds. The way back lasts the same time. Even if the program on the „Lady” already runs, it consequently takes up to 4/100 seconds till the answer is there. Within this time, the gentleman could send an amount of **SG** messages, which would be all transmitted to the „Lady“. For the first synchronisation that plays no role, but the surplus **SG** messages should remain saved in the receiver element in the lady program. At the beginning of the next step cycle, the „Lady” would then already have revceived an **SG** message without the „Gentleman” would have sent a new message. Then the “Ladies” would then no more wait for the „Gentlemen“. Therefore the „Gentleman” should wait for a sufficiently long time, before a message repeats.

ROBO-IF

ROBO-IF

The „Gentleman“ could naturally, also after transmitting, wait 1/10 seconds and then fisrt check whether he has received an answer. Waiting in a loop 10 times 1/100 second has however the advantage that the „Gentleman” can continue the program nearly immediately, if he has received the answer from the „Lady“. Now try out whether the program works better, if you change the Sync sub program in **TangoSyncGentl. rpp**, as described above. The robots should now always start as soon as the program on the second robot is started, independently of which robot is first started.

If you let your robots dance, until one of the accumulators becomes empty, the synchronisation is no more sufficient per step cycle. The robots synchronize in fact at the beginning of each step cycle, but during the cycle they run noticeably apart, if one of the accumulators comes to its limits. It is better to insert an additional synchronisation after each step. Here it is however clear that both programs run, so that you can give up a repetition. So that the initial synchronisation and the step synchronisation should not come in disorder, you should use for that two different sub programs **Sync1** and **Sync2**, which use different messages, for example **SH1**, **SL1** and **SH2**, **SL2**. You find the finished programs in the installation list of ROBOPro under

Sample programs\Manual\Tango Encoder Motor
Sample programs\Manual\Tango Pulse Switch

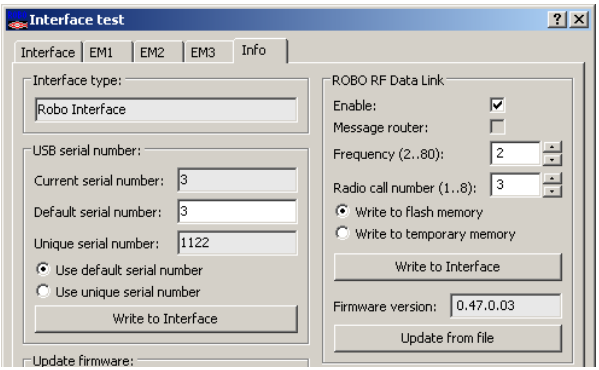
TangoGentl.rpp
TangoLady.rpp

You find there a TangoNachrichtenMonitor.rpp. If you have a third TX-Controller or a third interface with ROBO RF data link module, you can start this program in on-line mode, while both Tango robots dance. The program shows on the screen which messages were sent. The program uses level 3 elements and you need not understand for the moment how the program works.

4.5.1 Radio settings for the Robo interface


Interface test

Each ROBO interface gets an own radio call number between 1 and 8 and a frequency assigned to it, which you can adjust both in the **Info** window of the Interface Test. In the following illustration, you find the ROBO RF Data Link adjustments, on the right above. All RF Data Link modules, which should exchange messages with one another, must be adjusted to the same **frequency**. The frequency is entered as a number between 2 and 80. You can change the frequency, if several groups of robots in an area, for example in a school or with a competition, should communicate independently from one another. All robots, which belong to a group, use the same frequency. Different groups use different frequencies. You can also change the frequency, if the RF Data Link does not function well on the frequency used by you. Many radio systems, for example wireless PC networks, use the same frequency range (2,4 GHz) as the ROBO RF Data Link. If the RF DATA Link is disturbed by other radio systems, a frequency change can remedy the problems. Note however that you must then change all the RF Data Link Modules and the PC Raadio Module, since all devices in a group must always use the same frequency.



All the ROBO interfaces with RF Data Link, which are adjusted to the same frequency, must have a different **radio call number** between 1 and 8. The radio call number 0 is reserved for the PC Module of the RF Data Link, the „red box". Thus, maximaum 8 interfaces with instellaed radio module and an RF Data Link PC module can communicate with one another. The radio call number is, to say so, the telephone number of an interface in the radio net. You can assign the numbers 1 to 8 arbitrarily to up to 8 interfaces.

The **Enable** hook is nearly always set. However, you can deactivate a radio module in an interface, if you just do not use it in a model and you would like to save current without dismantling the module.

After you have made all the adjustments, you can save the adjustments in the interface with the **Write to interface** button. As a rule, you will write the adjustments in the **Flash memory**. The adjustments are then preserved, even if you switch the interface off. If you would only like to try something out only briefly, you can however write the adjustments into the **temporary memory** as well.

You need not care about the **firmware version**, that is the version of the internal control program for the RF Data Link. ROBOPro prompts you to automatically update the firmware, if that should be necessary.

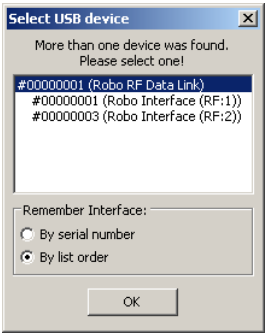
In order to make communication possible between the two Tango robots, first connect one of the two robots with the USB PC interface and open the interface test window. Possibly you must press the COM-USB button before and select the interface. In the interface test window, change to the partner info and adjust there the **frequency** 2 and the **radio call number** 1 and save the adjustments with the **Write to interface** button. Now close the interface test window, connect the other robot with the USB interface and adjust the **frequency** 2 and the **radio call number** 2.

Premises for the radio communication

Even if, as in the case of the two Tango robots, two interfaces change messages directly with one another over the RF Data Link modules, the red PC radio module is necessary. It serves as switching center for all the other modules and must therefore be connected to the USB interface of a PC. The PC must be switched on and may not change to a current saving or a sleep condition, so that the PC radio module should be supplied with current. The PC radio module must also be adjusted to the same frequency as the radio modules in the interfaces. The radio call number of the PC radio module is adjusted to 0 and cannot not be changed.

In order to adjust the frequency of the PC radio module, connect that PC module to the PC USB interface. You can then adjust the frequency over the Info Partner of the Interface test window exactly as in an Interface radio module . If the PC radio module cannot reach an interface with built-in radio module, for example because the frequency is not yet correct, you receive an error message when you open the Interface test the window. However, the error message only refers to the fact that no interface was found and that no inputs and outputs are thus available. ***However, you can make the adjustments in the Info window despite the error message***.

Interface selection through the interface (COM/USB) button



So far you probably worked mostly with only one ROBO interface. As soon as you have connected to the PC more than one ROBO interface or a PC radio module with more than one ROBO interface accessible over radio, the question arises, to which interface is to be connected the ROBOPro when a program in on-line mode is started, a program download is made or the Interface test window is opened. When you press the button for the interface options (COM-USB button), first the interface selection appears. When you select USB there and ROBOPro finds more than one interface at the USB bus or in the radio network, the selection window is indicated on the right. In this example, the PC radio module of a RF Data Link is connected to the USB interface of the PC. The PC radio module has found by radio two ROBO interfaces with RF Data Link, which have the radio call numbers 1 and 2. You can select in this window which of the two interfaces is to be used for future operations. As a rule, you select here one of the interfaces, and not the ROBO RF Data Link.

If you select the RF Data Link as in the picture above, ROBOPro connects to the interface with the smallest radio call number, 1 in the example. There is however an important difference: When the RF DataLink proper is selected, the adjustments in the Info Partner of the Interface test window refer to the PC radio module proper, and not to the interface radio module in the interface connected by radio. Thus you can change, for example, the frequency of both the interface radio modules and of the PC radio module by radio, without having to connect the interfaces to the PC over a USB cable. To that purpose, you first select, over the COM/USB button, one of the interfaces, change the frequency by means of the interface test the window and close again the interface test the window. If now you press again the COM/USB button, the interface with the changed frequency has disappeared from the list, since it can no more be achieved by the PC radio module. Select now the second interface and change the frequency. If you press once again the COM/USB button, probably no more selection list is indicated, because the PC radio module is the only accessible unit. If you have still connected further interfaces directly to USB and the selection list appears, select the PC radio module. When opening the interface test, now an error message appears that no interface was found. However, that does not disturb any more, since you would only like to change the adjustments of the Data Link. If you change the frequency of the PC radio module and press again the COM/USB button, the PC radio module and both the interfaces are indicated again, because now all the three have again the same frequency.

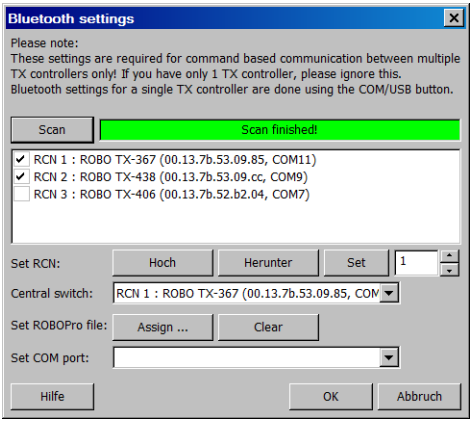
Why have you to change first the frequencies of the interfaces and only at last the frequency of the PC module? Try out what happened, if you first change the frequency of the PC module.

It can moreover happen that changes should only have an affect after a few seconds. If the selection list of the COM/USB button does not correspond to the current configuration, simply press the

COM/USB button once again. If you cannot any more reach an interface by radio, connect it at the best directly to the PC USB interface and control the RF DATA Link adjustments for the interface.

4.5.2 Bluetooth settings for the TX-Controller

Bluetooth settings



Every TX-Controller is given its own radio call number between 1 and 8, which you can allocate in the window for the Bluetooth settings. In ROBOPro programs, the radio call number is used like a telephone number to identify the individual controllers. First of all, you must log all interfaces onto the PC as Bluetooth modules. The instructions for the TX-Controller describe how to this†. To proceed, your PC needs either an integrated Bluetooth interface of a USB Bluetooth adapter. Once all controllers are logged onto the PC and switched on, open ROBO Pro. Press the COM/USB-Button and define one ROBO TX Controller which will be connected via USB or Bluetooth with ROBO Pro in online mode. Then open the Bluetooth settings window in ROBO Pro using the menu item **Edit Bluetooth** or press the button for the Bluetooth toolbar. Press **"Scan"** to fill the list with TX-Controllers.

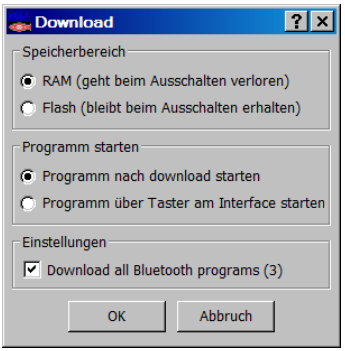
Depending on which Bluetooth adapter you use, it may happen that the interface names don't appear in the list. In this case, you can select the corresponding COM interfaces each in turn by pressing the COM/USB button and then doing an interface test to find out which interface is which. You also have to manually assign the corresponding COM to each TX controller under **Set COM Port**. But in most cases, this happens automatically.

The list with detected TX-Controllers is always sorted according to radio call number (RCN). You can change the radio call number by selecting a row in the list and pressing **up** or **down.** For example, if you select the row for radio call number 2 and press **up**, the controller moves one row upwards and is given the radio call number 1. The controller from row 1 moves one row downwards and is then given the radio call number 2. You can also select a row, enter the radio call number in the text box next to the **Set** button and then press **Set.**

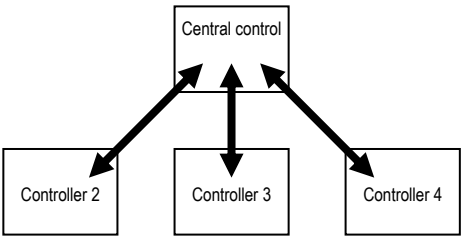
Once you have adjusted the radio call numbers, you then have to say which controller is to take part in the radio transmission. As a rule, that will be all the controllers in the list. However, in schools it is possible for controllers from other experimenting groups to appear in the list. To select the controllers for communicating with your program, place a tick at the start of the row in the list.

† For Bluetooth experts who can manage without instructions: the main code is 1234.

As it is tedious to download several programs on several interfaces, ROBOPro offers the possibility of allocating every TX-Controller a ROBOPro .rpp file in the list. Please use the **Assign** and **Clear** buttons. The download window offers the option of downloading all these programs at once.



Central control



The exchange of messages between 3 or more controllers uses a star topology. One of the controllers acts as **central control** as shown here on the left. For controller 2 to exchange messages with controller 3, the messages are sent via the central control. It is usually unimportant which controller assumes the central control function. However, it DOES matter which controller acts as central control for mobile robots that can move outside the radio range. This selection is again made in the window for Bluetooth settings. Another important point is that the online connection with the central control is more difficult and less reliable. This is because the PC plays the role of central control for the online connection, while the central controller acts as control centre for some Bluetooth connections, but not for others. As already indicated, this does actually work but it takes much longer to set up a connection.

Where are which settings saved?

The controller selection (tick at the start of the row) and the allocation of ROBOPro programs to controllers is part of a ROBOPro program. With multiple downloads, all programs assume the Bluetooth settings from the program that started the download. On the other hand, the allocation of radio call numbers and TX controllers is not saved in the ROBOPro program but on the PC. This makes it easier to swap ROBOPro programs with others.

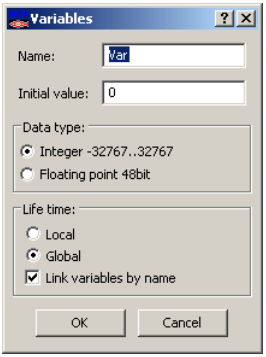
5 Level 3: Variables, panels & Co

Think about resetting ROBO Pro in the Level menu to Level 3 or higher!

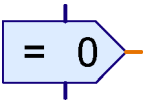
Just imagine that you discover a fascinating machine in a preciously unexplored side corridor of a museum, and you just have to emulate it in fischertechnik. But while investigating the machine you lose track of time, and don't notice that all the other visitors are leaving the museum. Only when the museum is already closed have you studied the machine sufficiently thoroughly to be able to make a replica. But unfortunately you must first spend an unpleasant night alone in the museum before you can set to work. So that this doesn't happen again, you go to the curator of the museum and volunteer to program a visitor counter, which will count all the visitors on the way in and on the way out again, and switch on a red warning lamp as long as there are still visitors in the museum. But how do you do that? How can you count something with ROBO Pro? The answer: with **variables**.

5.1 Variables and commands

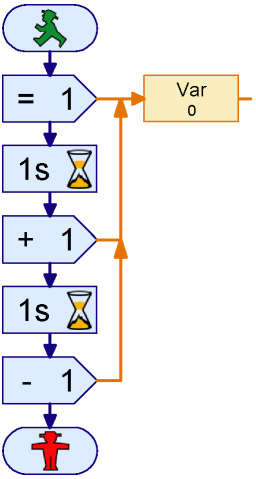
A variable is an element that can hold a number. In the variable Properties window you enter a **Name**, which should give some hint as to what sort of number is stored in the variable. Under **Initial value** you can specify what number should be stored in the variable at the beginning of the program. **Data type** determines whether the variable should be a whole number (for example, 1, 2 or 3) or a decimal, also called floating point number (for example, 1.3457). For now, we will only use whole numbers. The **Life time** setting will be explained in Section 8.4.2, *Local variables* on page 90.



You can alter the stored value by sending commands to the variable. A variable understands three different commands: **=**, **+** and **–**. The **=** command replaces the stored value with a new value. The **+** and **–** commands add something to or subtract something from the stored number.

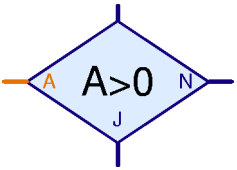


You send the commands to the variable with a ***Command element.*** Like most other program elements, the Command element has a blue program entry above and a blue program exit below. But to the right it has something quite new, an **orange** connection. That is a command output. Whenever the Command element is executed, it sends a command through this output to all elements connected to it. The variable has a corresponding command input on the left-hand side. When you connect the command output with the command input, instead of the usual blue connecting line, ROBO Pro draws an orange line. Program elements can send commands or messages over these orange lines, and thus exchange information.



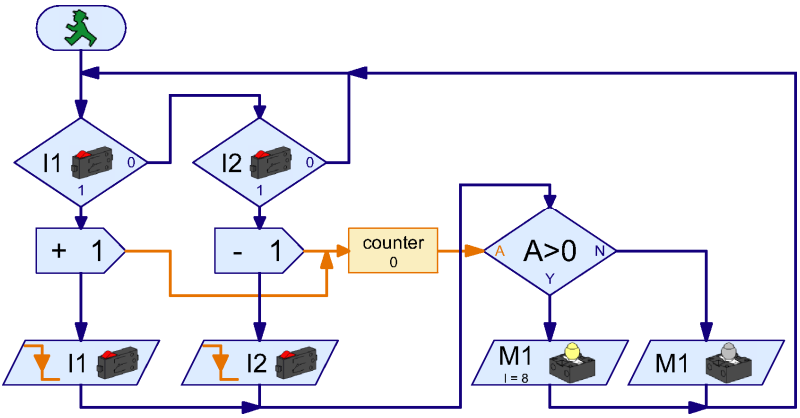
The program on the right initially sends the variable **Var** an **=1** command. As a rule, a command consists of the actual command, such as **=**, and a value such as **1**. The =1 command sets the variable to 1. After a second, the program sends the variable a **+1** command. The variable thereupon adds 1 to its previous value and now has the value 2. After a further second the program sends a **–1** command. Thereupon the variable has the value 1 again.

Now try to draw this simple program in ROBO Pro. You will find the command elements in the **Commands** group, the variables in the group **Variable, Timer, ...**. If you execute the program in online mode, you will see how the value of the variable changes.



That's all very well and good, you may be saying: I can look at the value of the variable, but just what do I do with it? Quite simple: The variable has an orange connection on the right, over which it sends messages with its current value to all connected elements. There are some elements in ROBO Pro with an orange input on the left, which you can link with the output of the variable. So, for example, in the group **Branch, Wait, ...** you will find a Yes / No Branch element which doesn't query an input directly, but rather can request any value at all, among others the value of a variable.

So the visitor counter for the museum can be programmed as follows:

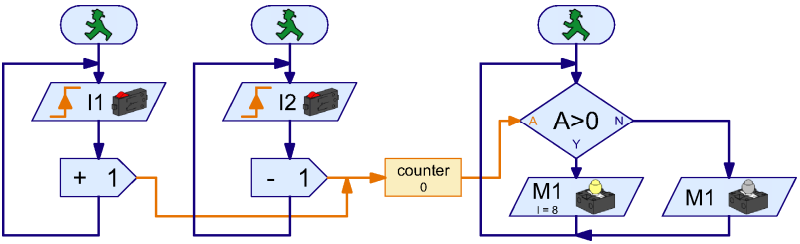


The entry turnstile operates the sensor on **I1**; the exit turnstile operates the sensor on **I2**. As soon as **I1** is pressed, the program sends a **+1** command to the variable **Counter.** Then the program waits until the sensor on **I1** is released again. With the sensor for the exit on **I2**, the behavior is exactly the same, except that here a **–1** command is sent to the variable **Counter**. Every time the counter changes, the state of the counter is checked. If the variable **Counter** has a value **>0**, the red warning lamp on **M1** is switched on; otherwise it's switched off.

Copy the above program and try it out. As soon as you press the sensor on I1 and release it again, the warning light on M1 lights up. If you operate the sensor on I2, it goes out again. If you operate I1 several times, you must operate I2 the same number of times to make the warning lamp go off again. Now try to see what happens if first 5 visitors come, then 2 go, then another 3 come. How many times do you have to operate the sensor on I2 to make the warning lamp go off again?

5.2 Variables and multiple processes

Perhaps you noticed while testing the visitor counter that problems arise if switches on I1 and I2 are pressed simultaneously. As long as one switch is pressed down, the program can't react to the other one. Since the visitors at the entrance and the exit may very well pass through the respective turnstiles at the same time, this leads to counting errors. You can avoid these errors by using several parallel processes. Up until now, all programs have had only one Start element. But there is nothing to stop you from using several Start elements. All program paths with their own Start element will then be worked through concurrently. So experts talk about ***concurrent processes***. Using this technique, you can change the visitor counter program as follows:



Now independent processes are used for I1 and I2. If the sensor on I1 is pressed, the process for I2 remains independent of this and can continue to monitor the sensor on I2. A separate process is also used to query count values and to switch the warning lamp on and off.

As you see, there is no problem about accessing a variable from several processes. You can send a variable commands from several processes and you can use the value of the variable in several processes. So variables are very well suited for exchanging information between processes.

The museum curator is so enthused by your brilliant visitor counter that he immediately asks you for the solution to another problem: The museum has installed a new exhibit. But as all the visitors want to see the new exhibit, there is such a crush there that nobody can see anything at all any more. So the curator would like to limit the number of visitors in the exhibition area to 10. The curator has installed a turnstile at the entry and exit respectively of the exhibit. The turnstile at the entry may be locked electronically. Now he only needs a competent program developer: you!

Try to develop the described program with ROBO Pro. Essentially, it functions like the visitor counter. You can simulate the electronic locking of the entry with a red lamp on M1, which should be switched on when there are 10 visitors in the exhibition.

5.3 Panels

After you solve the problem with the exhibit, the museum curator has yet another assignment for you. He would like to know how many people visit his museum in one day. Of course a program that can count is no problem for you, but how can you display the value? Of course, you could execute the program in online mode, which allows you to follow the values of variables. But for a computer-illiterate like the curator, that is rather complicated. Something simpler is required!

For cases like this, ROBO Pro has panels. A panel is a page of your own on which you can put displays and control buttons. Load your visitor counting program and, in the function bar, switch to **Panel**.

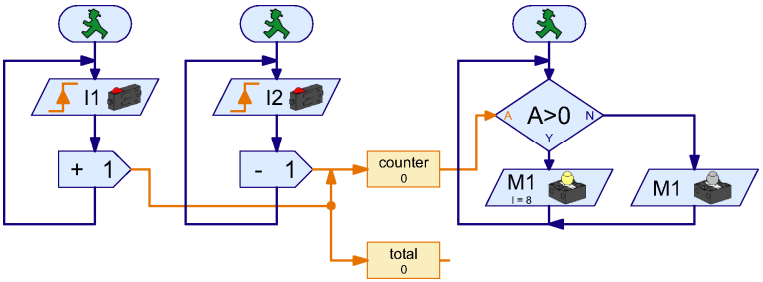
Initially, the control panel is an empty gray space. Onto this area you place displays and control elements which you find in the element group window und **Panel elements**. Among the panel elements you will find buttons, slider controls and the like. Under Displays you will find text displays, display lamps, and displays with rotary pointers.

Caution: A panel is part of a subprogram. If you have subprograms, make sure you create the panel under Main program and not under a subprogram! Later on, as a “pro”, you will be able to create multiple panels.

If you have drawn a panel and it has thereafter suddenly disappeared, then presumably you selected a subprogram in the subprogram bar. Switch back to Main program and your panel is sure to be there again.

For the visitor counter, you take a **Text display** (the color doesn’t matter) from the **Panel elements / displays** element window, and position it in the panel. This display is now required to show the number of visitors to the museum.

But first you must add to your program a second variable, which will count the number of visitors at the entrance without deducting them from the total again at the exit. In the function bar, you switch back to **Functions**, and insert the variable **Total** as follows:

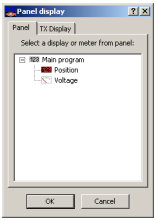


As you can see, a Command element can also be used to send a command to two variables at the same time. The variable **Total** does not receive the **–1** commands, because commands are only transmitted along the orange lines in the direction of the arrows. On the other hand, the **+1** commands are passed to both variables. But this is only done here as an example. As a rule it is simpler and more transparent to use a second Command element.

Tip: If orange lines branch, it is often more practical to draw the lines from their target to their origin. If you would like for example to draw the line to the variable Total, click first on the input to the variable Total and then move the line backwards to the branch point. If, on the other hand, you want to start an orange line on an existing orange line, you will have to double-click (with the left mouse button) on the point where the new line is to begin.

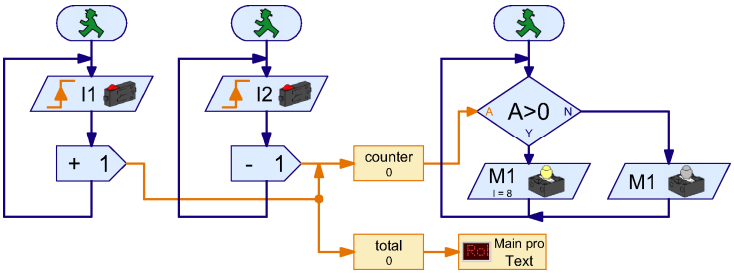
So, now you have a text display in the panel and a variable which you would like to present in the display. Now how do we link the two? As the text display and the variable are on separate pages, you would have trouble trying to connect the two with a line. For this reason there is a special element that transmits a value that is to be presented in a panel to the corresponding display. You will find the element **Panel output**, depicted above, at the end of the **Inputs, outputs** group. Insert one of these Panel output elements into your program next to the **Total** variable, and join the right-hand connection of the variable to the **Panel output**'s connection.

As you will normally have more than one display in a panel, you still need to let the panel output know which display to send the variable values to. This is done quite simply by means of the Properties window of the element. If you right-click on the Panel output element, you will see a list of all the displays that have been inserted so far into a panel. As every subprogram can have its own panel, the panels are listed according to subprogram. In our example, there are no subprograms, only the main program. Within this there is one display with the name **Text**. Select this display and click on OK.



As soon as you have linked the Panel output with a display, the symbol and the inscription change accordingly. The panel out we're using produces a connection with the text display named **Text** in the (sub)program **MAIN**.

Once you have inserted the panel output and linked it to the text display, the program looks like this:



Try it out straight away. As soon as you have started the program in online mode, the display in the panel shows the number of visitors that have passed through the turnstile at the entry.

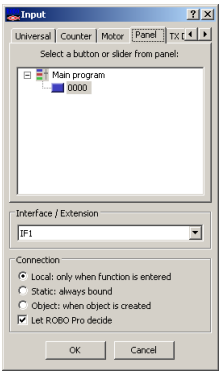
Hint: If you want to use more than one display in a panel, it is important that you give every display a different name, so that you can distinguish between them when linking them with the program. To do this, you right-click on the display in the panel. There, you can enter a name under **ID / Name**. Then, if you connect a panel output with the display, this name will appear in the selection window of the panel output. As we have only one display for the moment, however, the name is not important, and we retain the name Text.

The program is not quite perfect yet. What is still missing is a switch to reset the counter. For this purpose, however, we don't want to use a normal pushbutton switch, but rather a button we can push on the panel.

You will find this operating button in the element window under the group **Operating / Control elements**. In the function bar, switch to **Panel** and insert a button into your panel next to the text display. The inscription **Button** is of course not quite appropriate, but it can easily be change using the button's Properties window. Right-click on the button, enter for example 0000 as **Inscription** and confirm with **OK**.

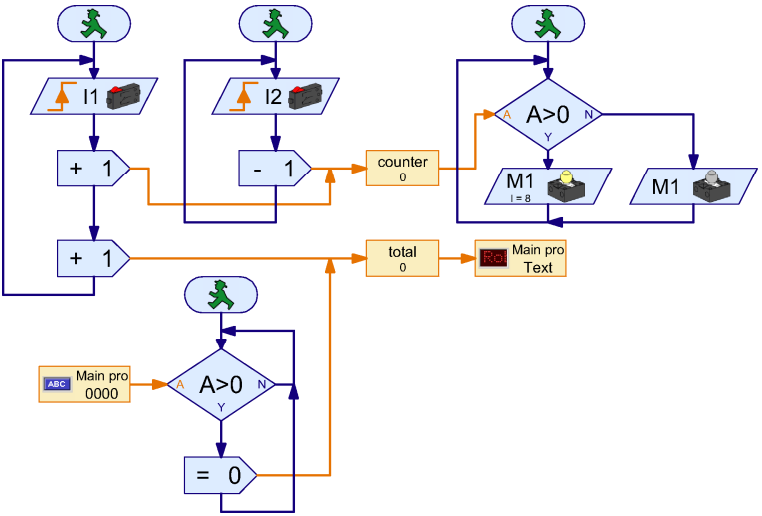
Exactly as in the case of the text display, we also need a program element that will link the button to the program. So start by switching back to **Function** in the function bar. You will find the illustrated **Panel input** element in the **Inputs, outputs** group in the element window. Position it in the flow chart below the existing program.

Now you still have to link the panel input with the button in the panel. To do this, you right-click on the Panel input element. As with displays, the control elements are listed according to subprogram, as every subprogram can have its own panel. Now select the **0000** button and confirm with **OK**.



You may have noticed that it is possible to set this element to all sorts of inputs through the tab bar of the Properties window. However, this will not be explained until the next section but one, *Command inputs for subprograms*.

The value delivered by a panel input is queried with a Branch element. You have already used this element to query variables. The complete program with the "set-to-zero" function now looks like this:



Whenever the **0000** button is pressed, an **=0** command is sent to the **Total** counter and sets the counter to zero.

5.4 Timers

After your triumphs, the museum curator does know what he could do without you, and so appoints you the museum's computer consultant. Of course, a position like this carries a lot of glory and renown with it, but also a lot of work, for example the following: The museum has many models that move when a button is pressed. But some visitors push for rather a long time on the buttons, so that the models overheat and keep needing to be sent off for repairs. Now the curator would the models to run for as long as the button is pressed, but only up to a maximum of 30 seconds at a time. Once the model has run, it should then take a pause of 15 seconds before it can be switched on again.

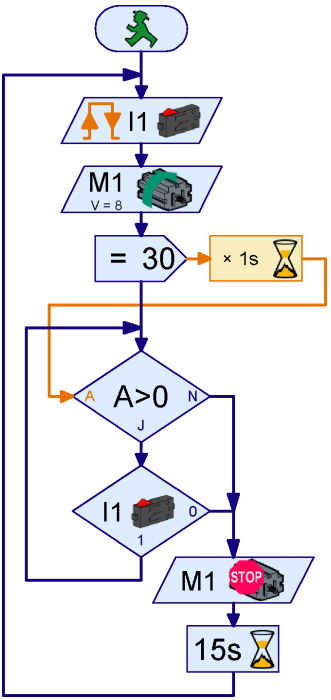
Hmm, no problem, you may be thinking. A few time delays, a few program branches, and you're done. Feel free to try it! After a while you will come to the conclusion that it is not so simple, and for two reasons:

- During the period of 30 seconds the program must query the button to establish whether the button is released before the 30 seconds expires. OK, granted, you can solve that with two concurrent processes, see Section 5.2 *Variables and multiple processes* on page 53.
- If a visitor releases the button after 5 seconds, and then presses it again after 15 seconds, the 30 second time delay must be started all over again. But the time delay has only been running for 5 + 15 = 20 seconds, and so is still active. Even with processes running in parallel, you can't start a time delay over again. Perhaps it would work in three processes with two time delays which you start in alternation, but thinking this through will bring on a headache.

Isn't there a simpler way to do this? Yes, there is: **timer variables**, or **timers** for short. Initially, a timer functions like a normal variable. The timer keeps track of a number and you can alter the number with **=**, **+** and **–** commands. What distinguishes a timer, however, is that it automatically counts down the number at regular intervals until it reaches 0. The time interval between decrements can be set in steps between one thousandth of a second and a minute. Many time control problems can be solved more elegantly with timers than with time delays. Do you see yet how you can solve the problem with a timer?

Correct: As soon as the visitor presses the button on **I1**, you start the model and then set the timer, using an **=** command, on 30 x 1 second = 30 seconds. Then you go into a loop, checking whether the period of 30 seconds has expired or whether the switch on **I1** has been released. When either of these stopping criteria is fulfilled, you stop the model and wait **15** seconds. Then it all starts again from the beginning.

Admittedly, the programs are starting to get more demanding. But just try to solve the following exercise: Develop a program with the same functionality but using time delays instead of timers! ***Note: This is a very difficult exercise and only intended for those who like to tinker around for a while longer with a puzzle every so often! Everyone else should simply proceed to the next Section.*** There are two approaches to solving this exercise: You can use two time delays which you start alternately in their own processes. As there is an off time of 15 seconds, one of the two time delays will have expired by the end of the second cycle at the latest, so that it can then be started over again. Another alternative would be to simulate a timer with a normal variable and a **Time delay** element with a short time delay of say one second.

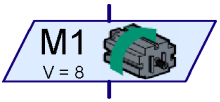


5.5 Command inputs for subprograms

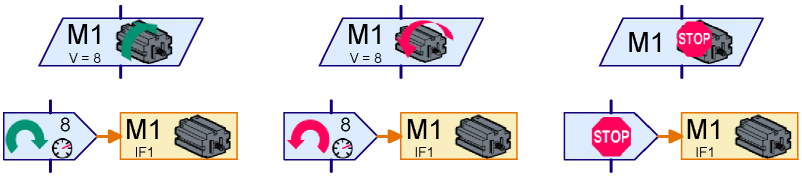
As always, your program works brilliantly, and fischertechnik is pleased, because all the models in the museum are being equipped with the ROBO Interface. Only, like public institutions everywhere, the museum is strapped for cash. So the curator would like to make do with as few Interfaces as possible. But then a ROBO Interface has four motor outputs and also enough inputs to control four models. As most models can only turn in one direction, you can control as many as 8 models via the single-pole outputs O1 to O8.

This of course saves the curator a lot of money. But on the other hand you now have to copy the program 7 times and adjust all the inputs and outputs to suit. Or maybe not? Couldn't you also do that with subprograms?

Indeed you could, but here a problem emerges: If you use the usual sensor queries from the **Basic elements** group in a subprogram, every call of the subprogram queries the same sensors and controls the same motors. The reason is that, in a Motor output element for example, the control command for the motor (right, left, or stop) and the motor output number (M1,...,M8) form a unit. As there is only one version of the subprogram, the same motor always appears in it. If you alter the motor number for one subprogram call, it will also be altered for all occurring calls of the subprogram. So, once again, you'd have to copy the subprogram 7 times, give every subprogram a different name and go all the way through manually adjusting the inputs and outputs.

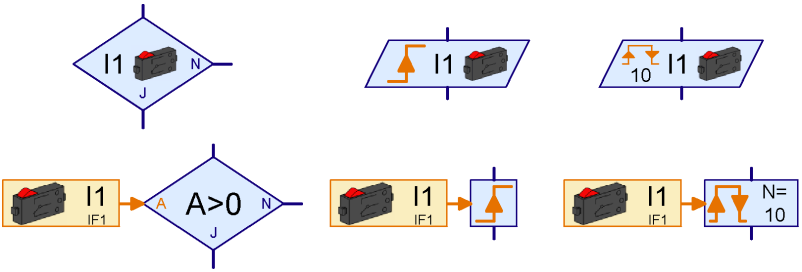


But there is a much more elegant solution to this problem. The trick is to separate the control commands from the motor symbols. Then you can put the control commands (left, right, stop) in the subprogram and the Motor elements in the main program. In the subprogram, using a Command element, which you have already encountered with variables, you then send the left, right or stop commands to the main program, where you can then dispatch them to the various motors. For a motor there is a Motor element that only represents a motor, without determining what the motor is to do. This element has a command input, to which you can send commands. You can replace elements from the **Basic elements** group with a Command element and a Motor element as follows:



In the upper row you see Motor elements from the **Basic elements** group. In the second row are depicted the corresponding combinations, achieving exactly the same effect, consisting of a command element from the **Commands** group with a motor element from the **Inputs, output** group. In fact, the upper elements are just abbreviations or simplifications for the combinations in the lower row. Each sends a left, right or stop command to motor **M1**.

The same also applies to querying sensors:



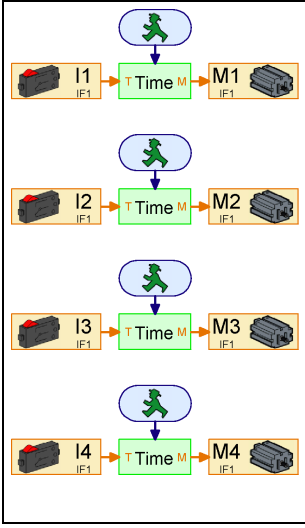
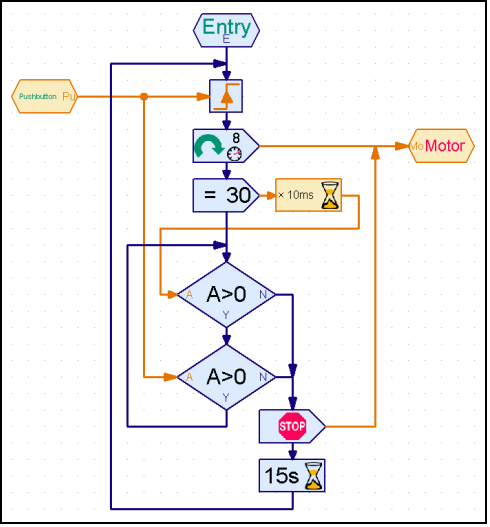
In the upper row, you see again elements from the **Basic elements** group. In the lower row you will find, for each of these basic elements, a corresponding combination of a digital input and an element from the group **Branch, Wait, ...**. You will find the orange **Digital input** element, like the Motor element, in the group **Inputs, outputs**.

Using this trick, you can separate the logic of a program from the inputs and outputs. But there is still something missing. If the motor and sensor elements are supposed to be in the main program and the commands in a subprogram, there must of course be a way of linking the sensor and motor elements with the subprogram. You will find the connection elements needed for this in the **Subprogram I/O** group.

Via a subprogram command input, you can send commands to a subprogram from outside. The Digital input element (sensor) sends its new value over the orange line if the state of the input changes (with what is known as an "= command"). In the element's dialog field you can give the input a name.

Via a subprogram command output you can send commands from a subprogram. So, for example, you can send the commands left, right, or stop from a subprogram to a motor. For this element too, you can enter a name in the dialog field.

Now you have everything you need for your ***multiple-model timer with subprograms***.

Main program	Subprogram **Time**
	

The **subprogram Time** is almost exactly the same as the program in the previous section. The **Wait for digital input** elements at the beginning and in the loop have, however, been replaced by **Wait for** elements, with data connections for orange lines, from the group **Branch, Wait, ...**. Both are linked to the subprogram command input **Sensor**. The two motor control elements at the beginning and end of the program have been replaced by command elements. Both send their commands to the subprogram command output **Motor**.

The subprogram **Time** is called four times in the **main program**. The subprogram command input **Sensor** has automatically generated the orange connection **S** on the left-hand side of the green subprogram symbol. The connection **M** on the right-hand side got there because of the subprogram command output **Motor**. The connection **S** of the subprogram symbol is connected each in case with one of the sensors **I1** to **I4** respectively. One of the motors **M1** to **M4** respectively is connected in each case to the connection **M**. In this way, each calling of the subprogram **Time** queries a different sensor and controls a different motor!

Try copying the above subprogram and main program and trying it out. You must draw the subprogram first, because otherwise you won't be able to insert the subprogram into the main program. If you have difficulties with the subprogram, refer once again to Chapter 4, *Level 2: Working with subprograms* on page 28.

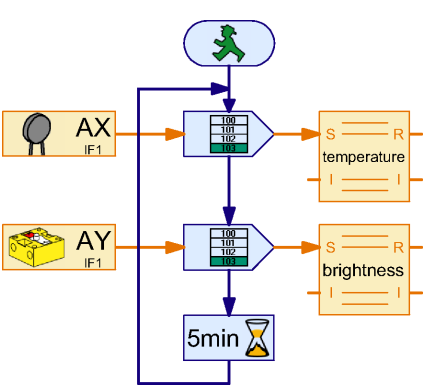
Note: You find further information to command inputs in the section:
6.3 *Sending arbitrary commands to sub programs* on page 68.

5.6 Lists (Arrays)

Now that all the trial equipment in the museum has been fitted with your cost-saving control system, there is not long to wait for the curator's next problem: In a space with very valuable antique exhibits, harmful temperature variations have been occurring recently. You presume that this has something to do with the level of insolation. To demonstrate this dependence, you would like to build a device which records the level of illumination and the temperature. Of course, the ROBO Interface has several analog inputs and you already know how to store values with the aid of variables. So the whole thing should be no problem, or should it? To record two values every five minutes over 12 hours requires 288 variables! But that would make for a gigantic and less than conspicuous program. Can we perhaps simplify this using subprograms again? We can, but there is a much better way. The **List** element (programmers call it an "array").

You can store not just one value but a whole list of values in a list. Initially, as a rule, a list is empty. If you send an **Append** command to the upper left data input marked **W**, the value specified in this command element will be appended to the end of the list. You can set the maximum length of the list between 1 and 32767 through the Properties window of the **List** element. This makes the program to record temperature and illumination quite simple:

The temperature sensor is connected to analog input **AX** and the brightness sensor to analog input **AY**. The program reads in both values every five minutes in a loop, and adds them to their respective lists with the **Append** command.

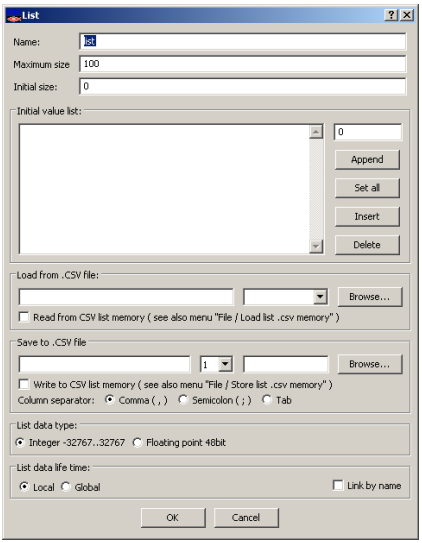


Hint: When inserting the command element you must activate the option Data input for command value in the Properties window. Then a data input will appear on the left of the command element, and you can connect the analog input to it.

To test the program, it is helpful to reduce the loop delay from 5 minutes to a few seconds.

You must now be wondering how you can read the stored values back from the list. There are two possibilities here: You can read the values as for an ordinary variable and process them further in your program. As the list contains more than one element, you first select the number of the element you want to retrieve at the left data input, marked **I**. Then the value of this element is given out at the data output **R** in the right-hand side.

But ROBO Pro can also store all the values from the list in a file on your computer, which you can then process further for example in Excel. As in the present case you only want to look at and compare the recorded illumination levels and temperatures, this is doubtless more practical. ROBO Pro saves the values in what is called a ***CSV file*** (comma-separated values). CSV files are text files which contain one or more columns each with a sequence of data. Thus you can also save several series of measurements such as temperature and illumination in separate columns of a CSV file. The columns are separated with commas. In countries where one writes 0,5 with a comma and not 0.5 with a period (e.g. Germany), a **semicolon (;)** is often used as the column separator. If you have problems exchanging CSV files between ROBO Pro and, for example, Microsoft Excel, you can change the **Column separator** in the Properties window of the list.



You can set the name of the CSV file and the column in which to store the contents of a list in the list's Properties window under **Save CSV file**. The data are saved when the program terminates in online mode, or, if you select the item **Save CSV files** in the **File** menu, while the program is still running (online or download mode). In download mode, you can separate the ROBO Interface from the PC for data recording and reconnect it for saving.

After you have executed the above program in online mode, you can open the .CSV file created from the data by ROBO Pro in Microsoft Excel or some other spreadsheet program. If you don't have a spreadsheet program, you can also use the Windows editor (Notepad), which you will usually find in the windows Start menu under Accessories.

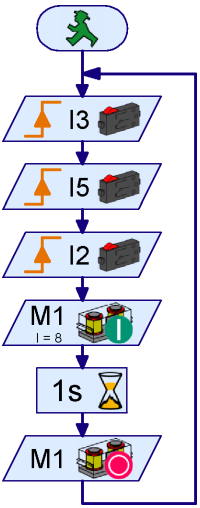
As long as the program is still running in online mode, you can also look at the data in a list by right-clicking on the List element.

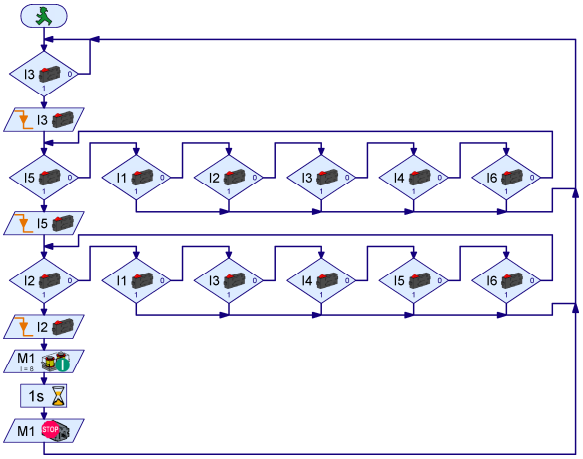
5.7 Operators

The illumination- and temperature-recording program worked well, but it became apparent from the recorded data that the temperature in the exhibition space of the museum has nothing to do with the sun. It has been established that some visitors have confused the airconditioning control in the exhibition space with a model control, and have been busily tinkering around with it. No wonder the temperature in the exhibition space has gone crazy!

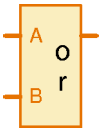
But this problem can be easily avoided with an electronic combination lock. The combination lock is to have a keypad with keys 1 to 6. If three figures are entered correctly one after another, the combination lock should release the climate-control cover by means of a magnet.

At first sight, such a lock is quite simple: The program simply waits until the right keys have been pressed in the right order. A program like this for the combination 3-5-2 can be seen on the right. But, on closer examination, this program has a problem. The lock can be quite easily picked, by pressing all keys from 1 to 6 three times in succession. In that way, the right key has always been pressed in every case. As Albert Einstein put it so aptly: “Things should be made as simple as possible—but no simpler.” So the program must enquire not only whether the right keys are pressed, but also whether any wrong keys are pressed. Now the program looks like this:

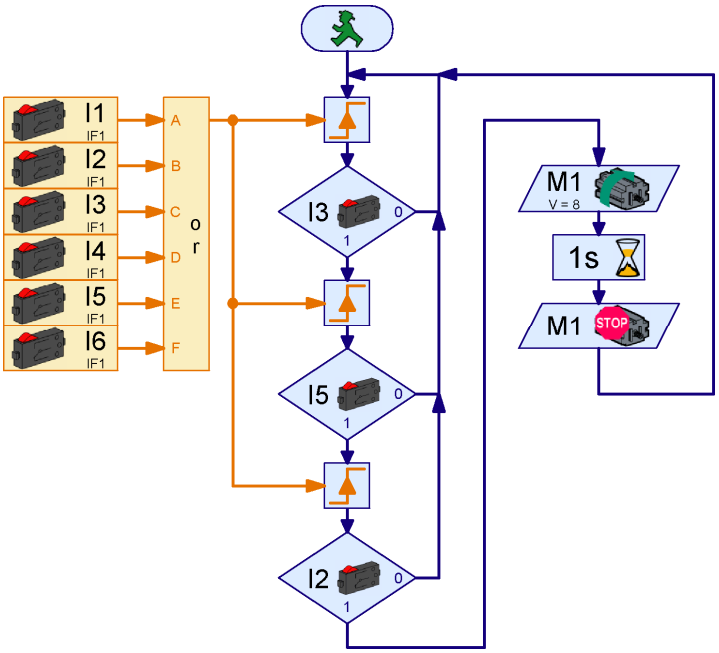




This program opens the lock only when the keys 3-5-2 are pressed without any other key being pressed in between. If for example the key 3 is pressed, the program first waits until the key is released again. If any key other than 5 is pressed next, the program starts again from the beginning. So the program works correctly, but it is neither simple nor conspicuous. Moreover, it is very difficult to change the code. But don’t worry; it can also be done simply and correctly, using ***operators***. There are various sorts of operators. You will find them under **Program elements** in the group **Operators.** For the combination lock, we first need an **OR operator**.



Several signals can be connected to the inputs of the **OR operator**. The operator always yields 1 whenever at least one of the inputs is 1 (or >0). If several pushbutton sensors are connected to the inputs of the **OR operator**, the output of the operator is always 1 when at least one of the buttons is pressed. The number of inputs can be set via the operator’s Properties window to up to 26. So all 6 keys can be connected to one operator. Perhaps you are asking yourself how we can use this to simplify the combination lock? Quite simple: with the operator you can initially wait in each step until any key is pressed. Then you can check whether it is the right key. Then you need 2 rather than 7 program elements per digit.

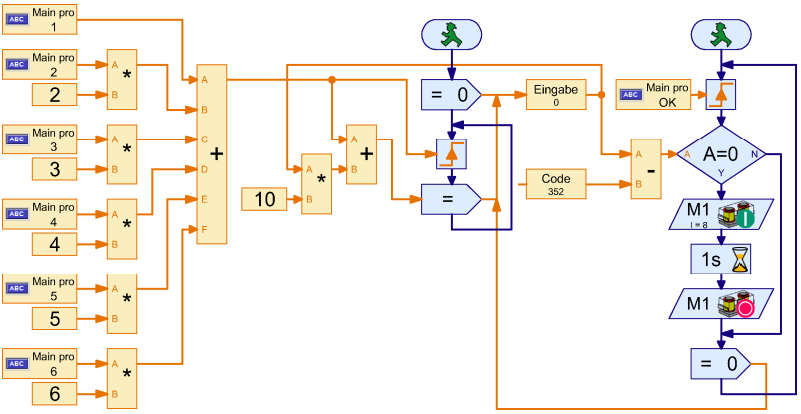


The buttons on inputs I1 to I6 are bundled together via an OR operator with 6 inputs. If at least one of the buttons is pressed, the OR operator yields an output value of 1; otherwise 0. With a **Wait for** element, the program waits until one of the buttons is pressed. Following this, we test immediately whether it was the right button. If so, we wait for another key to be pressed. If a wrong button was pressed, the program starts again from the beginning.

Alter the above program so that it uses panel elements in a panel instead of pushbutton sensors. Start by drawing a panel with 6 buttons marked 1 to 6. Then alter the digital inputs via the Properties window. You have to replace the branches by branches with data input and panel inputs.

The combination lock now functions without a hitch, but it is still not so easy to change the code (3 5 2). The inputs in three branch elements must be altered. It is not necessary to change the code regularly for the museum's airconditioning system, but if, for example, you were using the lock for an alarm system, you would presumably want to change the combination regularly. Of course, it would be easier if the code could be stored in a variable. The code could even be changed automatically. If, for example, a silent alarm is set off in the alarm system, the normal combination could be replaced by a special alarm combination.

In order to compare the combination variable with the input, you must also store the input itself in a variable. In the beginning the input variable should have the value 0. When you now press the 3 key, the variable should have a value of 3, with the next keystroke on the 5 key a value of 35, and finally after pressing the 2 key a value of 352.



The combination lock with code variable has two processes. In the process on the left, a number is assigned to each key with some times operators and a plus operator. The 1 key gets number 1, the 2 key number 2, and so on. The keys return a value of 0 or 1, and if you multiply this value by a fixed number X, a value of 0 or X results. As the values for unpressed keys are 0, you can add up all the values and end up with the numerical key value. As soon as a key is pressed, the input variable is set to 10 times the previous value plus the value of the key pressed. Multiplication by 10 shifts the existing value of the input variable one decimal place to the left (e.g. 35 becomes 350).

The process on the right waits until the OK key in the panel is pressed following input of the combination. The code variable Code, which has the value 352 if the code is correctly entered, is compared with the input variable. If they both have the same value, the opening magnet is activated, not otherwise. Finally the input variable is reset to 0. The variables Entry and Code are compared by comparing their difference with 0. You could also have use a Compare element.

If you press two leys at the same time, the values of the keys are added. So, for example, if you press 3 and 6 at the same time, the value 9 results. In this way you can build a super-secret lock, in which sometimes several keys must be pressed at the same time. Think which keys in which order you must press to open the lock with a code of 495. Don't forget that the **Wait for ...** element continues the program when the value increases, not only when it changes from 0 to 1.

Does the combination lock also work for 2- or 4-digit codes? If so, up to what number of digits does it work, and why? And what about the other combination lock programs?

6 Level 4: User defined commands

Remember to change ROBO Pro in the Menu **Level** to **level 4** (or higher)!

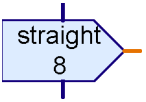
In the level 3, you were extensively engaed in how you can process Data by means of commands and you can, for example, steer motors. Thereby you used exclusively pre-defined commands like the = command or the right, left and stop command. In Level 4, sending commands over orange connections and using your own commands are now being linked with one another.

6.1 Processing of commands in a process

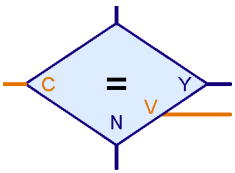
Surely, you have already written a program which controls a robot controlled via two wheels or a tracked vehicle. All the same whether the vehicle is to run left, right, straight or backwards, you must always give a right, left or stop command to two motors. And then you must always still note which motor drives the left and which drives the right wheel and whether the motor must turn to the left of or to the right, so that the vehicle drives forward. But a smart head as you has its head filled with other things, ingenious ideas for example, and therefore can not note such matters of minor importance.

Naturally, this problem can be solved by applying sub programs for each operation, but it would be still more elegant if you could write a sub program, which, like a motor output, has an information input, to which you then only need to send forward, backwards, left, right and stop commands and then it controls two motors correctly.

Now you will surely argue that in the command element of ROBOPro there is in fact an command to the left and one to the right, but no straight and no backwards command. But the good old command element is again and again good for a surprise. Create only for fun a new program, drag an arbitrary command element into the main program and simply enter once „Forward“ in the Characteristic window, under Command. And you will observe… it works!



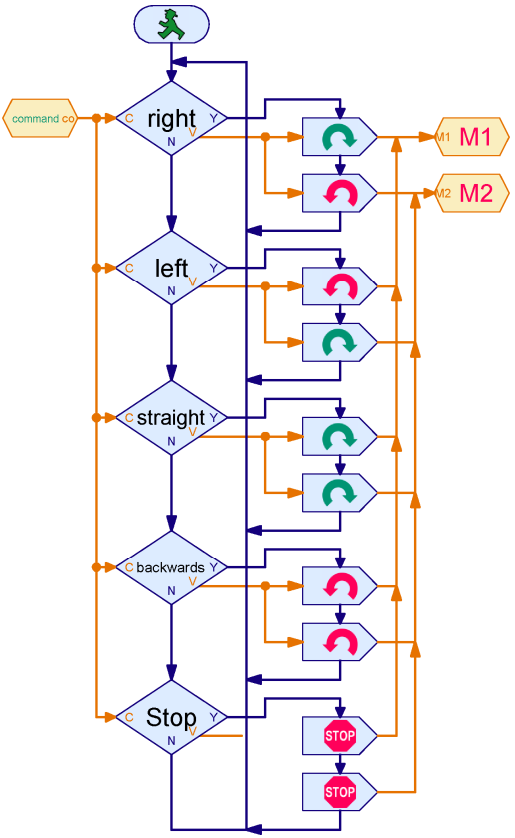
The next question is: what do you do with such an command element? Nevertheless, there is no element which can process such commands. If you send, for example, the forward command to a motor output and you try to start the program, ROBOPro will announce „No connected input can process the Forward command“. Starting from the Level 4, there are two new quite inconspicuous, however very efficient elements, which can process arbitrary commands: The „Command Wait" element and the Command filter. You find both elements at the end of the **Send, Receive** element group.



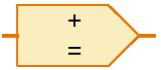
The task to develop a sub program, through which a 2-wheel robot may be controlled by means of the the forward, backwards, right and left commands, may be solved with both the elements. Let us try it first with the nearby „Command Wait“ element. You can send arbitrary commands to this element over the command input **C**. However, the element always only waits a completely determined command, which you can adjust. After the element has received this command, the program flux is branched to the **Y** output, otherwise to the **N** output. As you know, there is an command in ROBOPro, consisting of a name and a number, the command value. When the „Wait for Command“ element has received the command it waits, the command value is available at the **V** output.

Therewith it is quite simple to make the desired sub program. In a continuous loop, every of the 5 possible control commands is prompted with one „Wait Command“ element. After an appropriate command has been received, the command value is passed on to the right and to the left commands, which are sent to the two motor outputs, M1 and M2. If the sub program receives, for example, a **Vor** command with the value 8, the value 8 of the **V** output of the command filter is passed on to two command elements, which then send to the two motors a Right command with this value as speed. As a matter of fact, it is not so practical to use here only one single Right command element, which then sends commands to both the motors. With such constructions, it is often very difficult to hold apart the two motor command lines, so that then other commands should often go to both the engines.

In the example, the **V** output of the Stop command filter is not connected, because the Stop command elements need no value.

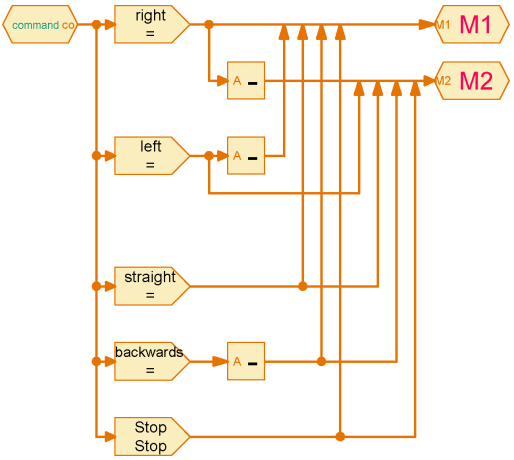


6.2 The command filter



The task in the previous section can also be solved with the command filter. With the command filter, you can, to say so, rename commands. When a certain command is sent to the left input, the element sends another command to the elements which are connected to the right output, but with the same command value as the command received at the input. Thus, you can make a motor control command, for example from one = command, such as Right or Left. With the Command filter, however, you can also convert, in particular, your own arbitrary commands to ROBOPro standard commands, so that you can release an action with your own commands.

In the illustration at the right, you see how you can configure the sub program for controlling a 2-wheel robot with the command filter. The top command filter, for example, converts the **cw** command to a **=** command. The motor outputs can also process = commands with a value from -8 to 8. Since with a clockwise rotation to the right of the model the two motors should turn into different directions, the value of the **=** command for the motor **M2** is made negative with one **-** operator. On the contrary, with the Left command, the value for the motor **M1** becomes negative. Forward and backwards are simpler, because both the engines turn into the same direction.



You need not neccessarily change the commands with the Command filter. The last command filter has the **Stop** command as both Input and Output command. With this element, the Stop commands are passed on directly to the motors. However, you need anyhow a command filter, so that other commands, as **cw** or **ccw**, should not be sent directly to the motor outputs.

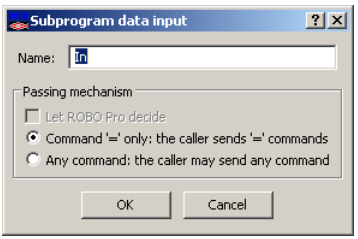
The great advantage of the command filter in relation to Wait at the Command element in the previous section is that you need not a process. That saves memory space and processing occurs immediately and not only with the next process change-over.

With the application of the command filter in this way, you must take care that you do not intermingle the Data lines for the two motors. It is easiest, if, as in the picture above, all the final arrows end on only one Data line. Sometimes it is also easier to use two command filters for an command, so that you have two separate outputs.

With the program above, the robot turns on the spot with the cw command. Try to change the program in such a way that M1 moves and M2 stands. You need for that two command filters for the Right command. One converts the command into one = command, the other one into a stop command. The command value is ignored with a stop command.

6.3 Sending arbitrary commands to sub programs

In the section 5.5 *Command inputs for subprograms* on page 58, you have already known command inputs for sub programs. However, you have copnnected there only digital or analogue input elements to the command inputs. Such elements always send one = command, if the value of the input changes. If you would liked to send another command to a command input of a sub program, you must indicate that in the Characteristic window of the command input. Starting from level 4, in the Characteristic window a new option,



Passing mechanism, was added.

If you select **Command ' =' only** here, you can only send = commands to the appropriate input of a sub program request. In addition, the last = command is **automatically repeated**, when the **sub program starts**. Otherwise, the sub program input would not have the correct value when the sub program starts. Just imagine that a digital input element is connected to the sub program input. These elements only send commands if the value at the input of the interface changes. If now the input is closed, the digital input element sends one = 1 command. When the sub program is started after the command was sent, it is important that the command is sent again, after the sub program is started. Otherwise the entrance would have a wrong value, until the element connected to the sub program input changes again its value.

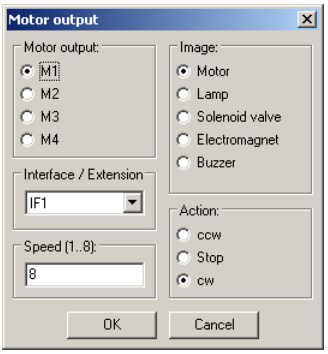
But this transmission automatic can also be disturbing and is rather unwanted with most commands. When you send, for example, one Start or a +1 command to a sub program, as a rule, you would not like that this should be repeated automatically. Therefore commands are not sent repeatedly, when you select the **Any commands** option.

Even if you send a **= command** to an **Any Command** input, the commands are **not repeated** with the sub program start. It may then occur that the value which the sub program input passes on should **not correspond** to the real value value at the input.

7 Controlling several Interfaces

One ROBO TX Controller is enough to control quite resource-intensive models. However, maybe some people like things a bit more extensive. If you can't get by with the existing inputs and outputs, you can ***connect*** up to 8 additional ***ROBO TX Controller*** to your ROBO TX Controller via the 6 pin extension connector. For those who still use the earlier ROBO Interface, there is also the possibility of controlling up to 3 ROBO Interfaces (each possibly with 3 I/O extensions) in online mode from your program.

7.1 Controlling Extensions



Perhaps you have already noticed the drop-down menu under **Interface / Extension** in the Properties windows for input and output elements. There you can select on which Interface or Extension Module an input or output is to be found. Provided you haven't made any other settings (see next Section), the list has the following entries:

- **IF1**: This is the ROBO TX Controller which can be connected to the PC as a so-called master.
- **EM1..EM3**: These are the ROBO TX Controller which are connected to the master as extensions.

So it is quite easy to control extension modules. You only need to choose the desired Controller (master or extension 1-8) for inputs and outputs. The operating manual of your ROBO TX Controller explains how to set up a ROBO TX Controller such that it will function as an extension.

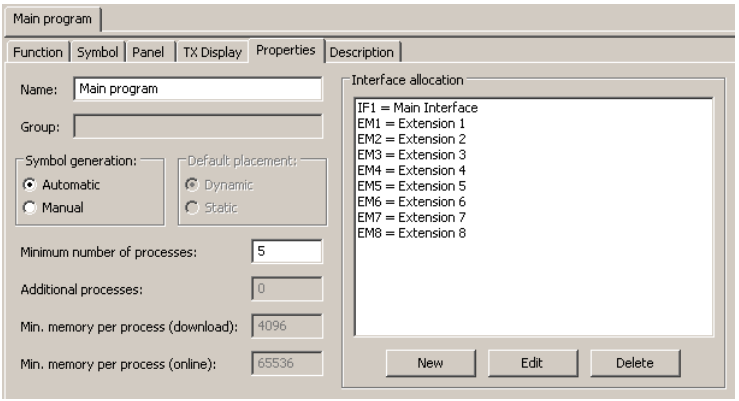
7.2 ROBO TX Controller and ROBO Interface together

If you want to control a ROBO TX Controller and a ROBO Interface simultaneously from one program, this only works in online mode. You can, for instance, connect a ROBO TX Controller with 8 extensions to a USB port. In addition, you can connect a ROBO Interface to COM1 or USB. This can include up to 3 ROBO I/O extensions. So that you can define the intended Interface in the Properties window of an input or output, you have to configure the Interface assignment.

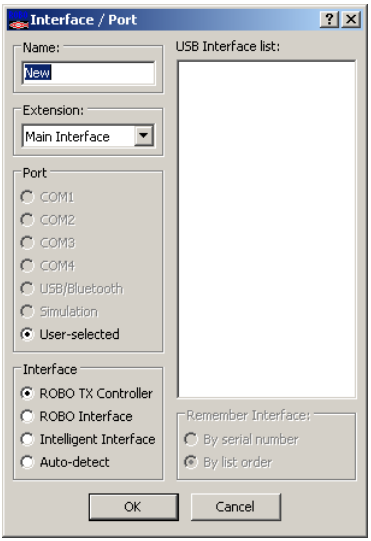
As long as you do not make different settings, you will find the entries **IF1**, **EM1-EM8** in the **Interface / Extension** drop-down menu. But you can add to or modify this list. There can be several reasons to do this:

- For greater comprehensibility you might want, rather than calling them IF1 or EM1, to give the modules names which specify which part of your machine or your robot the module is controlling.
- You might want to exchange two extension modules (e.g. EM1 and EM2), for ease of cabling, without changing your program.
- You might want to run a program, originally written for one ROBO TX Controller with more than 3 extensions, by using several ROBO Interfaces.

You can do all this quite easily by changing the **Interface assignment** in the Properties window of the ***main program***.



Here you can see which modules (master or extension) have been assigned to the names **IF1** to **EM8**. With the **New** button, you can add a new Interface. If you want to change an entry in the list, you select it and click on **Edit**. In either case, the following window is displayed:



COM/USB

- Under **Name** you can change the name used for the module. The name shouldn't be too long, because the space for the Interface name in the graphic symbol is very small. If you change this name, then you must usually also change the module name in all input and output elements that use this name.
- Under **Extension** you can specify whether the name refers to an Interface or to one of the extension modules 1 to 8.
- Under **Port** you can select the port to which the interface is connected. If you pick **User selection** here, the Interface used will be the one you selected in the toolbar under **COM/USB**. As long as you want to use only one ROBO TX Controller with several extension modules, this is the simplest, because this way someone else can use your program unaltered. If you are connecting additional ROBO Interfaces to your PC via USB, you specify here the port to which the relevant Interface is connected.
- Under **Interface** you can specify which Interface you would like to use. If you connect an earlier ROBO Interface or Intelligent Interface via a serial port, the program can detect automatically which type of Interface is involved (**Automatic** selection).
- The right-hand part of the window is only important if you have connected different Interfaces to the USB bus simultaneously. If, under **Port**, you click on **USB**, you can select one of the Interfaces under **USB Interface list**.

Caution: Unlike with the earlier ROBO Interface, only one ROBO TX Controller is connected to a PC via USB or Bluetooth. To this so-called master you can connect up to 8 ROBO TX Controllers as so-called extensions.

If you would like to operate more than one ROBO Interface on the USB bus, you must first assign to each Interface its own serial number. By default, all ROBO Interfaces were supplied with the same serial number, in order to void problems when exchanging Interfaces. The Windows operating system, however, only detects Interfaces with different serial numbers. You will learn more about this in Section 7.5 *Changing the ROBO Interface serial number* on page 73.

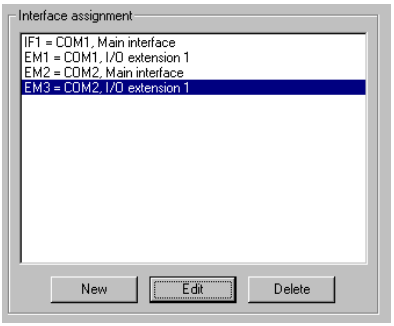
- Under **Remember Interface** you can specify how the program remembers the selected Interface. There are two possibilities here: If you select **By serial number**, the program stores the serial number of the ROBO Interface. Even if you connect other ROBO Interfaces to the USB bus and remove them, the program can always find the selected Interface again by means of the serial number. On the other hand, this has the disadvantage that the program now only works with an Interface with the same serial number. If you would like to use the program with an Interface with a different serial number, then you must change either the Interface assignment or the serial umber of the Interface. To get around problems with serial numbers, there is a second possibility: **By sequence**. If you select this item, the program stores the sequential order rather than the serial number. Although this can lead to confusion if you add or remove Interfaces on the USB bus, the program will run unaltered with any Interface.

7.3 Interface assignments in subprograms

Normally you will make all the Interface assignments for your program in the Properties window of the main program. However, you can also enter Interface assignments in subprograms. Then you can use the Interface assignments from both the main program and the subprogram in the subprogram. If two assignments have the same name, the assignment in the subprogram takes precedence. So, for example, you can define IF1 as accessing the main Interface in the main program, but standing for an extension module in a particular subprogram. This is very practical if you want to control a whole machine park, with every machine controlled by its own Interface. This way, you can develop the control programs for the individual machines as independent programs, with every main program accessing IF1. Later, you can install all the machine main programs as subprograms in one overall program. In the overall program you then need only modify the Interface assignments, but not the name in each individual input and output.

7.4 Tips and Tricks

If you want to run a program that was developed for a ROBO Interface with 3 extension modules on 2 Intelligent Interfaces each with an extension module, you can use the illustrated interface assignment. This replaces extension modules 2 and 3 with a further Intelligent Interface with extension module on COM2.



7.5 Changing the ROBO Interface serial number

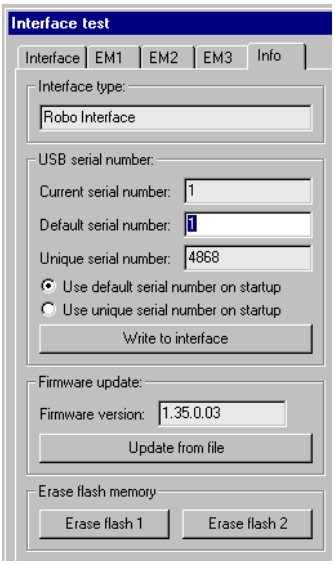
By default, all ROBO Interfaces and ROBO I/O-Extensions were supplied with the same serial number. As long as you only want to use one ROBO Interface on a computer, this is more practical, because in this way all Interfaces look the same to the computer, and there will be no problems with changing Interfaces. But if you want to operate more than one Interface on a computer via USB, you must alter the serial number of the Interface beforehand, so that the computer can distinguish between the Interfaces and address them. On the other hand, if you address the Interfaces via several serial ports, this is not necessary.

The procedure for changing the serial number of an Interface is as follows:

- Connect the Interface ***singly*** to the computer's USB bus.
- In the toolbar, switch to the programming environment for the ROBO Interface by pressing the button **Environment**
- Press the **COM/USB** button in the toolbar and select the USB port.
- Now open the Interface test window with the **Test** button on the toolbar and switch to the **Info** tab.

COM/USB


Test



- Under **Interface type** the type of Interface, so, e.g., **ROBO Interface** or **ROBO I/O Extension**, is displayed.
- Under **USB serial number** you can set the serial number used by the Interface at start-up. Every Interface has two built-in serial numbers, a **default serial number**, which is 1 as long as you do not set it to something else, and a **unique serial number**, which you can't reset and which is different for every Interface. The simplest way to use more than one Interface on the USB bus is to set the selection button for each Interface onto **Use unique serial number**. Then every Interface is guaranteed to have its own unmistakable serial number. If you use many Interfaces for one model, however, it can be very impractical to remember all the serial numbers. In this case it is simpler to set the default serial numbers of your Interfaces to, for example, 1, 2, 3, etc., and use these. After you have reset or selected the serial number, you still have to press the button **Write to Interface**. After changing the serial number, you must power down the Interface and reconnect it.

Caution: If the serial number is changed, the driver may have to be re-installed, which requires administrator privileges under Windows. If you change the serial number but can't re-install the driver, because you lack administrator privileges, you can no longer access the Interface via USB. In this case, you power down the Interface and hold down the Port button while powering up again. Then the Interface will start with the serial number 1, and will once again be recognized by the already installed driver. However, this does not reset the serial number permanently, i.e., on the next start-up without the Port button the previous serial number will be restored. To reset the serial number permanently, you proceed as described above.

- Finally, under **Update firmware**, you can update the internal control program of your ROBO Interface, if fischertechnik should ever offer a new version of the Interface firmware.

8 Program element overview

All the program elements available in ROBO Pro are arranged by element group in the following, and described in the order in which they are depicted in the element window.

8.1 Basic elements (Level 1)

8.1.1 Start

A process in a program always starts with a Start element. Without this program element at the beginning, a process is not executed. If a program contains several processes, each of these processes must begin with a Start component. The various processes are then started simultaneously.

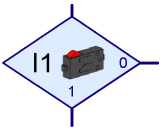
A start element has no properties that you can alter. For this reason, if you right-click on this element, unlike most other elements, ***no*** Properties window is opened.

8.1.2 End

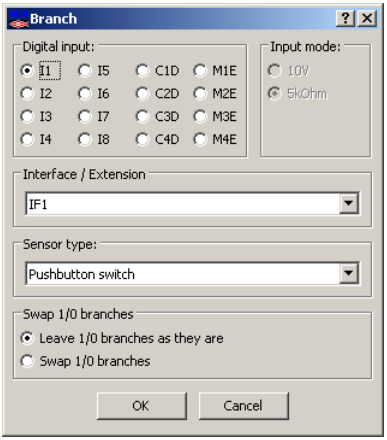
If a process is to be terminated, the exit of the last element is connected to an End element. A process can also be terminated at various different places with this element. There is also the possibility of linking the exits of different elements to a single End component. But is also quite possible that a process is executed as an endless loop and contains no End element.

The End element has no properties that you can alter. For this reason, if you right-click on this component, unlike most other elements, ***no*** Properties window is opened.

8.1.3 Digital Branch



With this Branch you can direct program control, according to the state of one of the digital inputs **I1** to **I8**, in one of two directions. If, for example, a sensor on the digital input is closed (=1), the program branches to the **1** exit. On the other hand, if the input is open (=0), the program branches to the **0** exit.

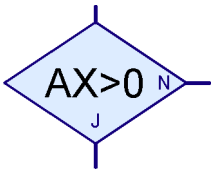


If you right-click on the element, the Properties window is displayed:

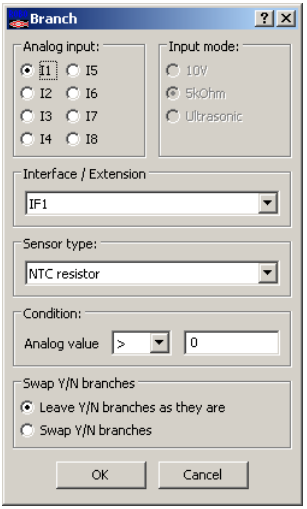
- Buttons **I1** to **I8** allow you to enter which of the universal inputs of the ROBO TX Controller is to be queried.
- Buttons **C1D-C4D** allow you to select one of the inputs C1-C4 of the ROBO TX Controller as simple digital input.

- Buttons **M1E-M4E** allow you to query one of these four internal ROBO Pro inputs. They are set to 1 as soon as a motor that is controlled by an ***Extended Motor Control*** element reaches a preset position.
- Under **Interface / Extension** you can select whether you want to use an input of the Interface or an input of an extension module. You can find out more about this in 7 *Controlling several Interfaces* on page 70.
- Under **Sensor type** you can select the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed contacts. ROBO Pro selects the **Input mode** of the universal input automatically according to the selected sensor. In Level 4 and above you can also select the **Input type** independent of the sensor.
- Under **Interchange 1/0 connections** you can interchange the positions of the 1 and 0 exits of the Branch. Normally the 1 exit is below and the 0 exit is on the right. But often it's more practical to have the 1 exit on the right. Press **Interchange 1/0 connections** and then the two connections will be interchanged as soon as you close the window with **OK**.

8.1.4 Analog Branch



As well as the digital inputs, the ROBO Interface has 6 Analog inputs: 2 resistance inputs AX and AY, two voltage inputs A1 and A1, as well as two inputs for distances sensors D1 and D2. With this Branch you can compare the value of an analog input with a fixed number and, according to the result of the comparison, branch to the Yes (Y) or No (N) exit.



If you right-click on the element, the Properties window is displayed:

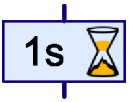
- Under **Analog input**, you can select which one of the universal inputs of the ROBO TX Controller is to be queried.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You can find further information about the various analog inputs in Chapter 7 *Controlling several Interfaces* on page 70.
- Unter **Sensor type** you select the sensor connected to the input. ROBO Pro selects the **Input mode** of the universal input automatically according to the selected sensor. In Level 4 and above you can also select the **Input type** independent of the sensor.

Sensor	Input mode	Displayed value
NTC resistor, photoresistor	Analog 5kOhm	0-5000 Ohm
Color sensor	Analog10V	0-10000 mV

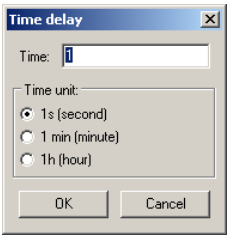
Ultrasound distance senor (Version TX, item number 133009 with 3 pin cable)	Distance	3-400 cm

- You can find further information about the various analog inputs in Section 8.7.2 *Counter input* on page 101.
- Under **Condition** you can select a comparison operator such as less than (<) or greater than (>) and enter the comparison value. The comparison value should lie in the range from 0 to 1023. When you start a program containing a Branch for analog inputs in online mode, the current analog value is displayed.
- Under **Interchange Y/N connections** you can exchange the position of the Y and N exits of the Branch. Normally the Yes (Y) exit is below and the No (N) exit is on the right. But often it's more practical to have the Yes exit on the right. Press **Interchange Y/N connections** and the Y and N connections are swapped as soon as you close the window with **OK**.

8.1.5 Time delay



With the **Time delay** element you can delay the continued execution of a process by a period you can set.

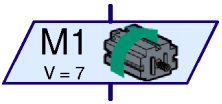


If you right-click on the element the Properties window is displayed. Here you can enter the time delay in seconds, minutes or hours. The time delay can be set over a range from one millisecond (that's one thousandth of a second) to 500 hours (that's just under three weeks). However, the time measurement becomes less accurate with longer time delays.

The following list shows the accuracy for various time delays.

Time delay	Accuracy
Up to 30 seconds	1/1000 second
Up to 5 minutes	1/100 second
Up to 50 minutes	1/10 second
Up to 8.3 hours	1 second
Up to 83 hours	10 second
Up to 500 hours	1 minute

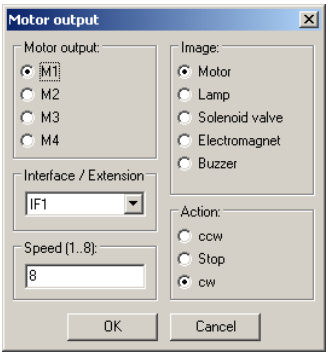
8.1.6 Motor output



With the program element **Motor output** you can switch one of the Interface's two-pole outputs **M1-M4**. The outputs from the Interface can be used for motors as well as for lamps or electromagnets. With a motor, you would like to be able to set the speed as well as the direc-

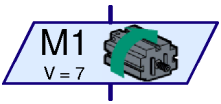
tion of rotation.

If you right-click on the element, the Properties window is displayed:

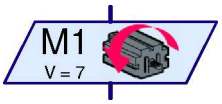


- Under **Motor output** you can set which of the four motor outputs **M1** to **M4** should be used.
- Under **Interface / Extension** you can select whether you want to use an output of the Interface or an output of an extension module or of another Interface. You can find out more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Image** you can select an image to represent the fischertechnik component connected to the output.
- Under **Action** you set how the output is to be affected. You can run a motor to the left (counterclockwise) or to the right (clockwise) or stop it. If you connect a lamp to a motor output (see tip under Lamp output), you can turn it on or off.
- Finally, you can specify a **Speed** or **Intensity** between 1 and 8. 8 is the greatest speed, brightness, or magnetic field strength; 1 the least. In the case of stopping or switching off, you naturally do not need to specify a speed.

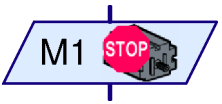
Here are listed some action symbols and images.



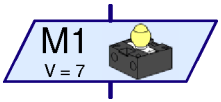
Motor right (cw)



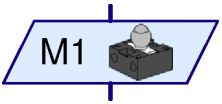
Motor left (ccw)



Stop motor



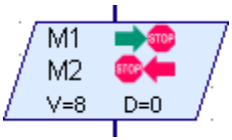
Lamp on



Lamp off

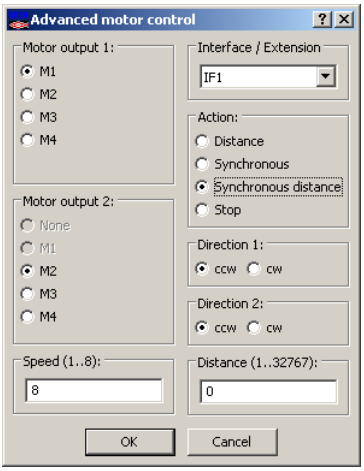
Tip: Sometimes even a motor is only operated in one direction, e.g. for a conveyor belt. In this case you can use a lamp output for the motor, so as to use one connection less.

8.1.7 Encoder Motor (Level 1)



The program element **Encoder Motor** is available from Level 1 on and allows comfortable control of motors with a built-in pulser (Encoder).

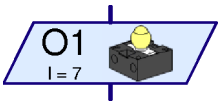
With this element you can either move a single motor a preset number of pulses or move two motors simultaneously, with or without specifying a distance. If you click on the element your left mouse button, the property window will be displayed:



- Under **Action** you select whether you would like to move one motor a specified distance (**Distance**), two motors with the same speed (**Synchron**) or two motors a specified distance with the same speed (**Synchron Distance**). To cancel any of these actions and to stop the motor, select the action **Stop**.
- Under **Motor output 1/2** you select the motor outputs which the action will affect. Depending on the action you can select one or two motors.
- Under **Interface / Extension** you can select whether you would like to use an output of the master or an output of an extension module. If the action controls two motors, both motor outputs must be of the same Interface. You will find more information in chapter 7 *Controlling several Interfaces* on page 70.
- Under **Direction 1/2** you set the direction in which the motors will move.
- Under **Speed** you enter the speed of the motors. If two motors are controlled, the speed of both motors is the same.
- Finally, under **Distance** you can enter the number of encoder pulses you would like the motor(s) to move.

You will find more information on using this element in section 11.6.1 *Encoder Motor (Level 1)* on page 126.

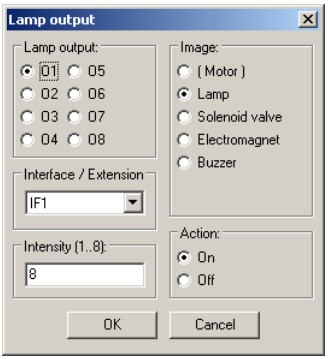
8.1.8 Lamp output (Level 2)



With the **Lamp output** program element you can switch one of the single-pole outputs **O1**-**O8** of the ROBO TX Controller. The outputs can be used either in pairs as motor outputs (see above) or individually as lamp outputs **O1**-**O8**. Unlike motor output, lamp outputs only take up one connection pin. That way you can control 8 lamps or solenoid valves separately. You connect the other lamp contact with the one of the ground sockets of the ROBO TX Controller (⊥).

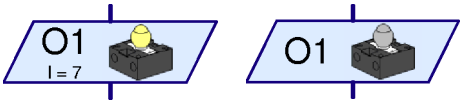
Tip: If you only wish to connect four lamps or motors, you can also use motor outputs for lamps. This is more practical, because in this way you can connect both lamp connections directly to the Interface output, rather than having to connect all the negative terminals separately to one of the ground sockets.

If you right-click on the element, the Properties window is displayed:



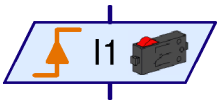
- Under **Lamp output** you can set which of the four motor outputs **O1** to **O8** should be used.
- Under **Interface / Extension** you can select whether you want to use an output of the Interface or an output of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Image** you can select an image to represent the fischertechnik component connected to the output.
- Under **Action** you set how the output is to be affected. You can switch a lamp on or off.
- Finally, you can also specify an **Intensity** between 1 and 8. 8 is the greatest brightness; 1 the least. In the case of switching off, you naturally do not need to specify an intensity.

Here are listed the symbols for the various actions for the **Lamp** image.

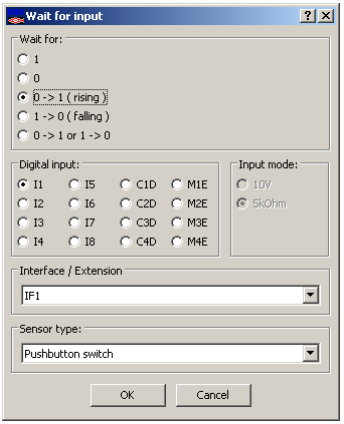


Lamp on Lamp off

8.1.9 Wait for input



The **Wait for Input** element waits until one of the Interface's inputs is in a particular state or until it changes in a particular way.

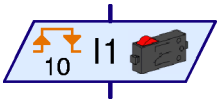


If you right-click on the element, the Properties window is displayed:

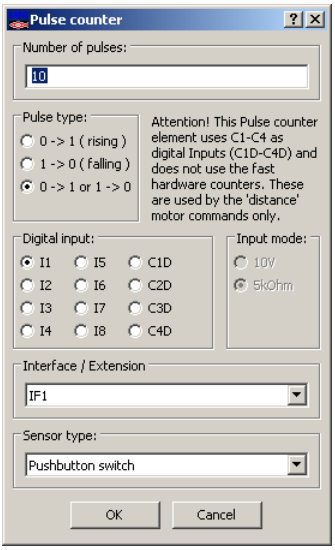
- Under **Wait for** you can select the type of change or the state to be waited for. If you select **1** or **0**, the element waits until the input is closed (1) or open (0). If you choose **0 -> 1** or **1 -> 0**, the element waits until the state of the input ***changes*** from open to closed (0->1) or from closed to open (1->0). In the last case, the element waits until the state of the input changes, regardless of whether it's from open to closed or vice versa. To help you understand this further, it is explained in Section 3.6 *Other program elements* on page 22 how you can emulate this element with the Branch element.

- Under **Digital input** you may enter which one of the inputs is to be queried. You can select one of the universal inputs **I1-I8**. The other inputs are described in section 8.3.1 *Digital Branch* on page 75.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You can find out more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Sensor type** you can select the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed contacts. ROBO Pro selects the **Input mode** of the universal input automatically according to the selected sensor. In Level 4 and above you can also select the **Input type** independent of the sensor.

8.1.10 Pulse counter



Many fischertechnik model robots also use pulse wheels. These gear wheels operate a sensor four times for every revolution. With these pulse wheels you can turn a motor on for a precisely defined number of revolutions rather than for a given time. To do this, you need to count the number of pulses at an input of the Interface. For this purpose there is the **Pulse counter** element, which waits for a user-definable number of pulses.



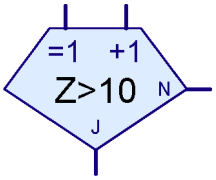
Note: There is a spezial element to control **encoder motors** which slows down the motors in time and therefore works more precisely. See section 8.1.7 *Encoder Motor (Level 1)* on page 78.

If you right-click on the element, the Properties window is displayed:

- Under **Pulse type** you can select the type of pulse to be counted. If you choose **0 -> 1** (rising), the element waits until the state of the input has changed from open to closed (0->1) the number of times you have specified under **Number of pulses**. If you choose 1 -> 0 (falling), the element waits until the state of the input changes from closed to open (1->0) the specified number of times. With pulse wheels, however, the third possibility is used more often. Here the element counts both **0 -> 1** and **1 -> 0** changes, so that 8 pulses are counted per revolution of a pulse wheel.
- Under **Digital input** you may enter which one of the inputs is to be queried. You can select one of the universal inputs **I1-I8**. **C1D-C4D** selects one of the counter inputs. However, this does not make use of a fast hardware counter. Nevertheless, the maximum count frequency is several 100 Hz.

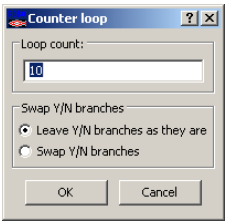
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Sensor type** you can select the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed contacts. ROBO Pro selects the **Input mode** of the universal input automatically according to the selected sensor. In Level 4 and above you can also select the **Input type** independent of the sensor.

8.1.11 Counter loop



With the **Counter loop** element you can very easily have a part of the program executed several times. The Counter loop element has a built-in counter. If the counter loop is entered via the **=1** entry, the counter is set to 1. If the counter loop is entered via the **+1** entry, 1 is added to the counter. According to whether the counter is greater than a value you have prescribed, the counter loop branches to the Yes (**Y**) or No (**N**) exit. You will find an example for this in Section 3.6.4 *Counter loop* on page 24.

If you right-click on the element, the Properties window is displayed:

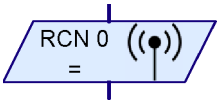


- Under **Number of iterations** you enter the number of times the counter loop is to be exited via the No (**N**) exit before the Yes (**Y**) exit is activated. The value you enter should be positive.
- If you click on **Interchange Y/N** connections, the **Y** and **N** connections will be changed over as soon as you close the window with OK. According to where the **Y** and **N** connections are, the program section to be repeated will be to the right of or under the counter loop.

8.2 Send, Receive (Level 2-4)

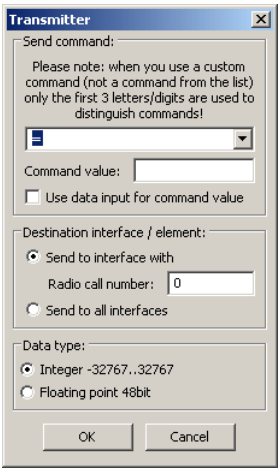
In this element group, you find program items which you can use for sending and receiving messages over the ROBO RF DATA Link or over the serial interface of the ROBO interface.

8.2.1 Sender (Level2)



With the transmitter you can send a command or, fairly generally, a message, via Bluetooth (in case of the ROBO Interface via the ROBO RF Data Link) to another interface. In this way, for example, several robots can communicate with one another.

Properties window for the transmitter element

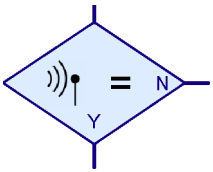


- Under **Send command**, you can enter the command and starting from level 3 the Command value. An command consists of a name and a numerical value. The numerical value can also be determined over a Data input. With command names, which are not selected from the list, only the first 3 letters or numbers are considered. You can indicate more than three characters, but „Hello“, „Help“ and „helicopter“ stand all for the same message, because all begin with ‚Hel'. Large and lower case and special characters (blank!? , % and the like) are likewise not differentiated. XY! And XY? stand also for the same message. Numbers are however differentiated, so that XY1 and XY2 are different messages.
- Under **Destination interface/element**, you can select to which interface or program element must be sent the command. In most cases, you will send an command to an interface with a certain **radio call number** or to **all interfaces**. Starting from level 4, there is additionally the possibility to use a **group** between 10 and 255 as receiver address. Commands to a group are not sent to a certain interface with a certain radio call number, but to receiver elements in which the same group number is indicated. Thus you can, for example, differentiate from whom a message was sent, by using another group for each transmitter. The group numbers begin with 10, because the numbers 0 to 9 are reserved. Your learn more about the reserved group numbers with the Receiver element.
- Under **transmit channel**, you adjust how the message should be transmitted purely technically. In the level 2, this selection is not available and it is always transmitted by **RF** (not including self). By RF means that the message is sent via Bluetooth (ROBO RF Data Link). If **RF (including self)** is selected, the message is also sent to the interface which has sent the message. In order that it functions, under Destination interface/element a destination must be selected which includes the sending interface, consequently, for example, to **all interfaces** or to a **group of receivers**. You can also send a message only to the sending interface it**self**. You can use this function, for example, for communication between different processes. Starting from level 4, the ROBO Interface also supports sending commands over the serial **COM interface** of the ROBO interface. For this purpose, you must connect two interfaces with a serial null-modem cable.
- Under **optimization** (starting from level 4) you can adjust whether identical commands should be sent several times. With many commands, it does not make a difference whether you send the command once or several times after one another. Without optimization, for example, commands sent several times after one another fill the transmission buffer of RF Data Link, so that other commands can not be sent so quickly. Therefore it is reasonable to delete identical commands. As a rule, you will be willing to **delete** an command **if** only it is **identical to the last buffered command**. If you send in this mode, for example, 2x Start and then 2x Stop, only 1x Start and then 1x Stop is transmitted. If you however send Start, Stop, Start and Stop fast after one another, consequently not two times the same command after one another, the commands are transmitted unchanged. However, you can also indicate that commands should to be deleted if they are **identical to any buffered command**. On the contrary, with many commands the optimization is not reasonable and the **normal** mode should be used. An example for that is the **Add** command, with which you can add elements

to a list. With a list, it finally makes a difference whether an element 1x or 2x is added. In the level 2, the option **delete if identical to the last buffered command** is always selected.

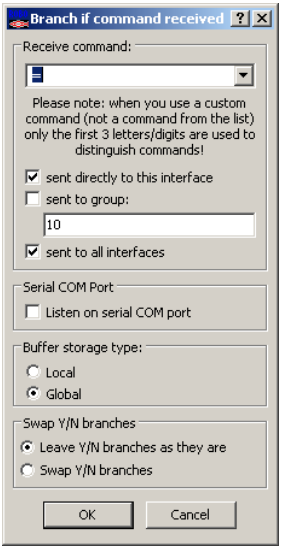
- Under **Data type** you can select whether the value of the sent command is a whole number or a floating point number. Also see chapter 12 *Working with decimals* on page 130.

8.2.2 Receiver (Branch when command is received, Level 2)



This element is the counterpart to the preceding transmitter element. Depending on whether a certain command was received or not, the element is branched to the J or to the N output.

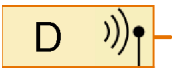
Properties window for the Receiver element



- Under **Receive command**, you enter the command which the receiver should receive. As already explained with the transmitter, only the first 3 letters or numbers of the name are considered. Then you must still select whether the receiver only reacts to commands which were sent directly to the interface, consequently with a certain radio call number, or to commands which were sent to all interfaces. You can also select both. As already described with the transmitter, starting from the level 4, you can send messages also to a certain group. Such messages are received by all receiver elements with which this group was indicated. The groups 10 to 255 can be used arbitrarily. The groups 0 to 8 correspond to the radio call numbers 0 to 8. The group 9 is reserved for sending to all interfaces. When sending messages, it does not make a difference whether you send to the group 1 or to the radio call number 1. But for receiving you can not indicate a radio call number, because each interface knows its own radio call number. By indicating a group of receivers from 1 to 8 for the receiver, you can however receive messages, which were actually meant for another interface. But groups of receivers smaller than 10 you can only use starting from level 5.
- Under **Serial COM port** (starting from level 4, only ROBO Interface), you can indicate that the element can receive also messages from the COM interface. Here it is a matter of a global adjustment whether the COM interface should be activated or not. When in a program one single transmitter or receiver uses the COM interface, all receiver elements can receive messages from the COM interface.
- Under **Type of buffer** you can specify whether the memory area in which received commands are stored is ***local*** or ***global***. If you select global, the element can also receive commands when the subprogram in which the element lies is not active.

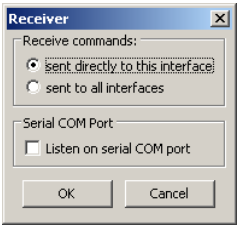
- Under swap **Y/N branches**, you can interchange the position of the Y and N outputs of the branching. Normally, (Y) is the Yes output down and the No (N) output on the right. Often it is however more practically if the Yes output is on the right. **Press the swap Y/N branches**, then the Y and N connections are exchanged, as soon as you close the window with OK.

8.2.3 Receiver (Level 3)



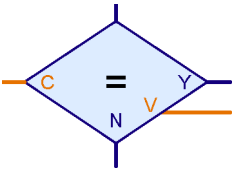
The receiver element described in the previous section is mainly meant for the level 2, since it can only receive commands, but no command values. The level 3 receiver element receives, on the contrary, arbitrary commands with command value. You do not indicate with this element any command which the receiver element should receive. The element sends fairly easily all the received commands to the elements connected to the output.

Properties window for the Transmitter element



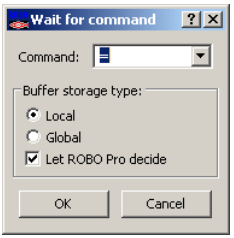
- Under **Receive command**, you indicate whether the receiver only reacts to commands which were sent directly to the interface, consequently with a certain radio call number, or to commands which were sent to all interfaces. Starting from the level 4, you can also select a certain group. You learn more about groups of receivers in the two previous sections *Sender (Level2)* and *Receiver (Branch when command is received, Level 2)*. Elsewise than with the level 2 Transmitter, with the level 3 transmitter, you can select only one option. You can however connect the outputs of two or several receiver elements with different selection, if you want to receive commands with different destination Data. In particular, you can also switch so receivers with different groups.
- Under **Serial COM port** (starting from level 4), you can indicate that the element can also receive messages from the COM interface. See thereto the previous section *Receiver (Branch when command is received, Level 2)*.

8.2.4 Wait for command (Level 4)



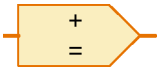
The **Wait for Command** element is used similarly as the *Receiver (Branch when command is received, Level 2)*, for waiting a command. However, it does not wait for commands which are sent by ROBO RF Data Link or other interfaces, but for commands which are sent to the command input on the left side of the element. If you connect a *Receiver (Level 3)* element there, a level 2 receiver results. However, this element has additionally a Data output on the right side. Always when an command has been received and so the program flux is led to the J output, the numerical value sent with the command is available at the command value output W. Since the W and J output belong together, you cannot exchange the J and N outputs for this element. You find an example in the section 6.1 on page 66.

Properties window for the „Wait for command" element



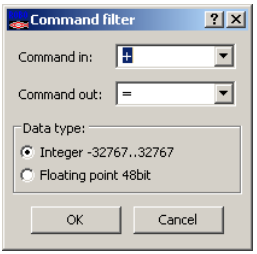
- Under **Command**, you select the command for which the element should wait. You can also enter an own command, whereat however only first 3 letters or numbers are considered. See for that the description under Send command in the section 8.2.1 *Sender (Level2)* on page 82.
- The **buffer size** of the command buffer is only indicated starting from level 5. As the level 2 receiver, this element notes how many commands have been received. Since waiting at the command element must also note the command value for each command, the maximum number of commands is here however limited. For usual applications, the standard value of 4 commands should be completely sufficient, since a program command received is mostly, as much as possible, processed immediately.

8.2.5 Command Filter (Level 4)



With the command filter, you can, so to say, re-label commands. When a certain command is sent to the left oputlet, the element sends another command to the elements which are connected to the right output, but with the same command value as the command received at the output. Thus you can make, for example, a motor command command from one = command, such as Right or Left. You may find an example in the section 6.2 on page 67.

Properties window for the Command filter element



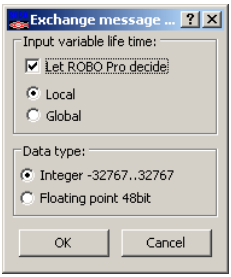
In the Command filter element you select two commands: the command which is expected at the input and the command converted into this command and sent to the output. You can also enter your own command, whereby however only the first 3 letters or numbers are considered. See thereto the description under Send command in the section 8.2.1 *Sender (Level2)* on page 82.

Under **Data type** you can select whether the value of the command (sent or received) is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.

8.2.6 Exchange Message (Level 4)

Similar to the way the command filter exchanges the name of a command (see previous section), this element exchanges the value. In conjunction with the command filter this allows you to take one message and generate several different messages with different values. For example, if you would like to program a control for a track vehicle so it will understand commands like “left”, “right” or “forward”, you can convert the command “left“ into a “=“ command to the motor element using the command filter. In addition, with this element you can replace the value of the “=” command by 0 or the negative value and send it to the other motor. The command whose value is to be exchanged is sent to the **B** input. You place the new value in the **W** input.

Property window for the Exchange message element



- Under **Input variable life time** you can select whether the W input stores the value in a local or global variable.
- Under **Data type** you can select whether the value of the command (sent or received) is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.

8.3 Subprogram I/O (Level 2-3)

In this element group you will find program elements that you only need for subprograms.

8.3.1 Subprogram entry (Level 2)

A subprogram can have one or more Subprogram entries. The main program or the higher-level subprogram passes control to the subprogram via these entries. In the subprogram's green symbol that is inserted into the higher-level program, one connecting pin for each Subprogram entry is inserted on the upper side. The connections on the symbol have the same sequence (left to right) as the Subprogram entries in the subprogram's functional plan. If you right-click on the element the Properties window is displayed. There you can give the entry a name, which will then be displayed in the symbol. You can find out more about subprograms in Chapter 4: *Level 2: Working with subprograms* on page 28.

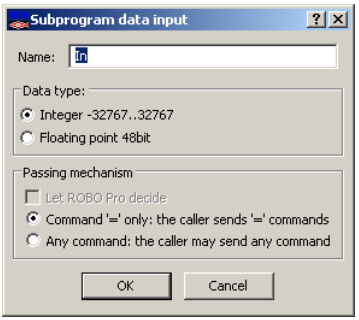
8.3.2 Subprogram exit (Level 2)

A subprogram can have one or more Subprogram exits. The subprogram passes the control back to the main program or higher-level subprogram via these exits. In the subprogram's green symbol that is inserted into the higher-level program, one connecting pin for each Subprogram exit is inserted on the lower side. The connections on the symbol have the same sequence (left to right) as the Subprogram exits in the subprogram's functional plan. If you right-click on the element the Properties window is displayed. There you can give the exit a name, which will then be displayed in the symbol. You can find out more about subprograms in Chapter 4: *Level 2: Working with subprograms* on page 28.

8.3.3 Subprogram command input (Level 3)

Via this element, subprograms can be linked to input elements such as switches in the main program or higher-level subprogram, or supplied from there with values from variable elements, e.g. co-ordinates. In the subprogram's green symbol that is inserted into the higher-level program, one connecting pin for each Subprogram command input is inserted on the left side. The connections on the symbol have the same sequence (top to bottom) as the Subprogram command inputs in the subprogram's functional plan. There is a thorough explanation of the use of this element in Section 5.5 Command inputs for subprograms on page 58.

Property window

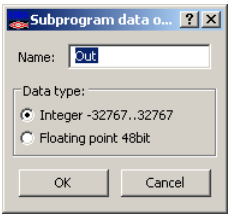


- Under **Name** you can enter the name for the command input. Only the first two characters are displayed in the green subprogram symbol.
- Under **Data type** you can select whether the value of the received command is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.
- Under **Passing mechanism** (Level 4 and above) you can select whether the input accepts ***only "=" commands*** or arbitrary commands. If variables or Interface inputs are connected to the input in the subprogram call, you should select only "=" commands. In this case the subprogram input stores the lastly transmitted value, making the correct value available immediately when the subprogram is started. If you select arbitrary commands, you can also send commands like stop or own commands to the input. These commands will only be forwarded to the subprogram if the subprogram is active. This makes sense when the subprogram contains a motor element, for example, and you would like to send commands to this element from outside. You will find more information in section 6.3 *Sending arbitrary commands to subprograms* on page 68.

8.3.4 Subprogram command output (Level 3)

Via this element commands such as left, right, stop can be sent to motors or other output elements in the main program or in the higher-level subprogram. In the subprogram's green symbol that is inserted into the higher-level program, one connecting pin for each Subprogram command input is inserted on the right side. The connections on the symbol have the same sequence (top to bottom) as the Subprogram command inputs in the subprogram's functional plan. There is a thorough explanation of the use of this element in Section 5.5 Command inputs for subprograms on page 58.

Property window

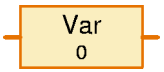


- Under **Name** you can enter the name for the command output. Only the first two characters are displayed in the green subprogram symbol.
- Under **Data type** you can select whether the value of the sent command is a whole number or a floating point number. Also 12 *Working with decimals* on page 130.

8.4 Variable, List, ... (Level 3)

Program elements in this group can store one or more numerical values. They allow you to develop programs with a memory.

8.4.1 Variable (global)



A variable can store an individual numerical value between -32767 and 32767. The value of the variable is set by connecting an **=** Command element to the command input on the left-hand side (see Section 8.5.1 = *(Assignment)* on page 94). Via the Properties window, one can also give the variable an initial value, which the variable will retain until it receives the first command altering the value.

ROBO Pro creates only one variable for all variable elements with ***the same name*** and **Variable type** = **Global**. All global variables with the same name are identical and always have the same value, even if they occur in different subprograms. When one of these variable elements is altered via a command, all other variables with the same name are changed too. There are also local variables (see next section) to which this doesn't apply.

As well as the **=** command, a variable also understands **+** and **–** commands. So, for example, if a variable receives the command **+ 5**, it adds the value 5 to its current value. In the case of the **–** command, the value communicated with the command is subtracted from the variable's current value.

Caution:
If in a **+**or **–** command the value of a variable goes outside the allowable range of values, 65536 is added to or subtracted from the value of the variable to bring it back in the valid range. As this behavior is normally unwelcome, you should make sure that this does not happen.

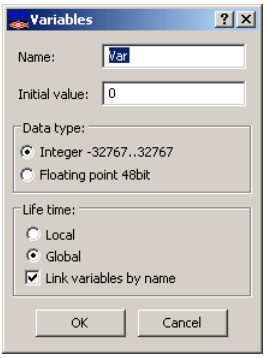
Every time the value of the variable changes, it sends an **=** command with the new value to all elements connected to the command output of the variable. If you want to monitor the value of a variable, you can connect a panel display to the output of the variable (see section 8.7.6 *Panel input* on page 104).

Here is a compendium of all the commands that the Variable element can process.

Command	Value	Action
=	-32767 to 32767	Sets the value of the variable to the value passed with the command.
+	-32767 to 32767	Adds the value passed with the command to the current value of the variable.
-	-32767 to 32767	Subtracts the value passed with the command from the current value of the variable.

Incidentally, the odd value range of -32767 to 32767 results from computers calculating in the binary system, and not in the decimal system as we do. In the binary system 32767 is a round number, a bit like 9999 in the decimal system. But we don't need to worry about this, as the computer converts all the numbers from the binary to the decimal system. We only notice anything in the maximum values and when there is an overflow in calculations.

Properties window for variables.



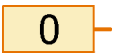
- Under **Name** you can enter a name for the variable.
- Under **Initial value** you can enter an initial value for the variable. The variable retains this value until it gets a new value via an **=**, **+**, or **–** command.
- Under **Data type** you can select whether the value of the variable is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.
- The **Life time** item is only significant for variables in subprograms, and is more precisely explained in the following section, "Local variables". In the case of variables in the main program, both settings have the same effect.

8.4.2 Local variables

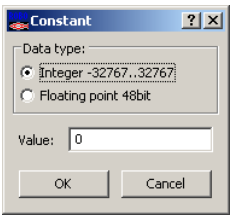
All **global** variable elements with the same name use one and the same variable and always have the same value. That is presumably what you expect and what is generally practical. But if you use variables in subprograms, that can lead to big problems. If your program has more than one parallel process, multiple instances of a subprogram can be being executed at a time. In this kind of situation, it usually leads to chaos if the program uses the same variables in all processes. For this reason there are ***local variables***. A local variable behaves almost exactly like a global variable, with one difference: the local variable is only valid in the subprogram in which it is defined. Even if two local variables in separate subprograms have the same name, they are distinct, independent variables. Even if one program is being executed in parallel by several processes, the subprogram in each process has an independent set of local variables. Local variables exist only as long as the subprogram in which they are defined is being executed. Therefore local variables are not assigned their initial values at program start, but rather every time the relevant subprogram is started. As a subprogram is supposed to do the same thing every time if it is called more than once, it is much more practical if the variables are set to their initial values at each call. Local variables have, so to speak, no memory of previous calls of the same subprogram.

In the main program, local and global variables behave in the same way, as the overall program and the main program are started at the same time. However, local variable are somewhat more efficient in program execution. On the other hand, list elements should rather be defined globally, because the storage area for global variables is bigger than for local variables.

8.4.3 Constant



Like a variable, a constant has a value, but this value cannot be altered by the program You can link a constant with a data input of a subprogram symbol, if the subprogram is to use the same value at all times. Constants are also very practical for calculations with operators. You will find an example of this at the end of Section 5.7 *Operators* on page 62.



Property window for constant

- Under **Data type** you can select whether the value of the

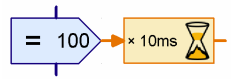
constant is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.

- Under **Value** you enter the value of the constant.

8.4.4 Timer variable

A timer variable behaves essentially just like a variable. Even the distinction between normal and static variables exists with timer variables. The only difference is that a timer variable counts down the stored value at fixed time intervals until it reaches 0. Once the timer value reaches zero, it stays there. If the timer value becomes negative, e.g. through a minus command, the value returns to 0 at the next time step.

The rate at which a timer variable counts down can be set between 1/1000 second per step and 1 minute per step in the Properties window. In doing so, you should observe that the accuracy of the timer depends on the time steps set. If, for example you set a time on 1 x 10 seconds, the next time step can take place a short time later (e.g. as soon as one second), or not until 10 seconds later. So timers are only as precise as the time steps set. Therefore, you should prefer to select small time steps, for example 10 x 1 second or 100 x 0.1 seconds rather than 1 x 10 seconds. You should only select a time step of a minute if the program is to wait at least an hour. Then, one minute more or less is not going to make much difference.

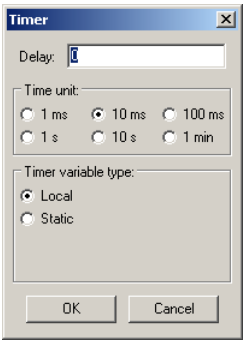


The number of steps to be counted down is generally assigned to the timer via an **=** command from a command element. In the example illustrated, 100 steps of 10ms each are counted down. This corresponds to a duration of 1000ms=1 second. The precision of this is 10ms.

Timer variables enable you to solve even difficult time measurement and delay problems easily. For example, if a robot is to discontinue a search after 20 seconds, you can set a timer variable on 20 x 1 seconds (or 200 x 0.1s) at the beginning of the search, and then query regularly in the search program whether the timer value is still greater than 0. You can also reset the timer to its starting value where there is partial success in the search.

If you want to measure a time, the timer variable should initially be set to the biggest possible positive value (30000 or 32767), so that there is a lot of time left before the timer value reaches 0. If you want to know how much time has passed since then, you subtract the current timer value from the initial value.

Properties window for timer variables



- Under **Delay** you can determine an initial value for the timer variable. As a rule, you will enter 0 here, and set the value of the timer variable with an = command at the appropriate time. But if the timer is supposed to start running at the start of the program or of a subprogram, the corresponding value can be entered here.
- Under **Time unit** you can select the size of the time steps at which the timer variable will be counted down.
- Under **Timer variable type** you can set whether the timer is a global or a local variable (see Section 8.4.2 *Local variables* on page 90).

8.4.5 List

The **List** element corresponds to a variable in which one may store not just one but several values. The maximum number of values that can be stored in a variable is determined in the Properties window.

You can append values to the end of the list or remove values at the end of the list. You can also change or read any value in the list or exchange any value in the list with the first value in the list. A value cannot be inserted in the middle or the beginning of the list directly. But you can write an appropriate subprogram that will perform these functions.

The following functions of a list are used by sending commands to the **W** (for write) input. The following commands can be sent to the **W** input:

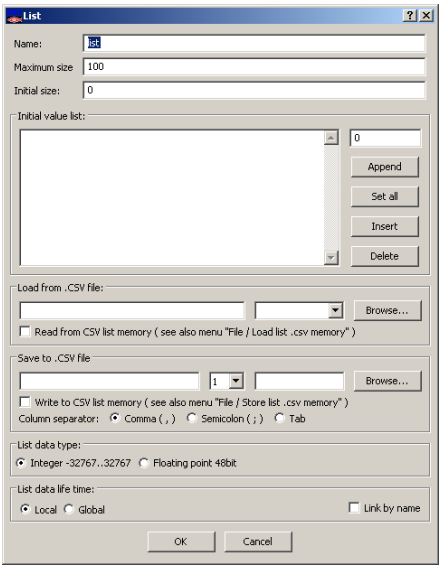
Command	Value	Action
Append 	-32767 to 32767	Appends the value passed with the command to the end of the list. The list gets bigger by one element. If the list already is at its maximum size, the command is ignored.
Delete 	0 to 32767	Deletes the given number of elements from the end of the list. The value communicated with the command is the number of elements to be deleted. If the number is greater than the number of elements in the list, all elements are deleted. If the number is 0 or negative, the command is ignored.
Exchange 	0 to 32767	Exchanges the given element with the first element in the list. The value passed with the command is the position number of the element to be exchanged.

Via the **I** (for Index) input, a specific element of the list can be selected. To do this, you send an **=** command to the **I** input with the desired element number. The first element is element number 0. Another value can be assigned to the element selected via the **I** input by sending an **=** command with the desired value to the **W** input.

The element selected via the **I** input can be queried via the **R** (readout) output. If the **I** input, or the value of the entry selected by the **I** input, changes, the list sends the current value of the selected entry to those elements connected to the **R** output.

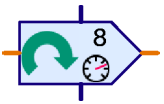
Via the **I** output you can query whether the index defined at the **I** input is valid. If N is the number of elements, a value between 0 and N-1 must be present at the **I** input. If this is the case, the **I** output sends an **=** command with the value N, in any other case with value 0, to all connected elements.

Properties window for lists



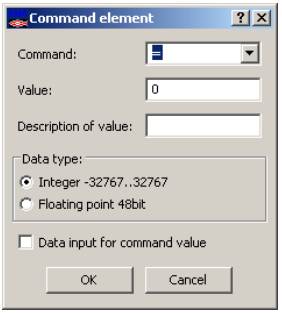
- Under **Name** you can enter a name for the list.
- Under **Maximum size** you can enter the maximum number of elements on the list. This size cannot be exceeded by **Append** commands.
- Under **Initial size** you enter the number of elements with which the list is to be initialized at start time.
- Under **List of initial values** you can enter the initial values to be pre-assigned to the list. With the buttons to the right of the list you can edit the list.
- Under **Load from .CSV file** you can select an Excel-compatible .CSV file from which the list should take its values. In the selection field in the middle you can choose the column of the .CSV file to be used for this purpose. The file is loaded straight away and displayed under **List of initial values.** When you start the program or perform a download, ROBO Pro will try once more to load the current values from the file. If this is not successful, the values stored under List of initial values are used.
- Under **Save to .CSV file** you can specify a file to which the contents of the list should be saved after the program ends. This works, however, only in online mode and only for static lists (see next point). The contents of the list are written in the selected column of the file. Under **Column separator** you can select whether the individual columns in the list should be separated with commas or semicolons. In countries where 0.5 is written with a period, a comma should normally used as the column separator. As people write 0,5 with a comma in Germany, in Germany a semicolon is also often used as a column separator. If you have problems importing a ROBO Pro CSV file into, for example, Microsoft Excel, try a different column separator.
- Under ***List data type*** you can select whether the list contains whole numbers or floating point numbers. Also see 12 *Working with decimals* on page 130.
- Under **List data life time** you can set whether the list elements are global or local variables (see Section 8.4.2 *Local variables* on page 90). For large lists (with a maximum size over 100 elements) type **Global** is to be recommended, because more memory is available for global variables than for local variables.

8.5 Commands (Level 3)



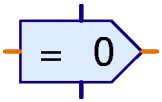
All program elements in this group are command elements. Depending on the application they may also referred to as message elements. When the command element is executed (i.e. when the flow of control passes into the blue entry at the top of the element), the command element sends a command or a message to the element connected to the output on its right. There are various commands like right, left or stop, which have different effects on the connected element. As a rule, the connected elements understand only a few commands. The commands each program element understands and the effects of these commands are listed alongside the various program elements. Most commands are also accompanied by a value. With a **right** command for example, one specifies also a speed between 1 and 8. A **stop** command, on the other hand, has no additional value.

Properties window for command elements



- Under **Command** you can select the desired command from a list of all possible commands.
- Under **Value** you enter the numerical value that should be passed with the command. If no value is to be passed, this field remains empty.
- Under **Value description** you can enter a short indicative text (e.g. X= or T=), which will be displayed in the command element with the value. The text should make clear what sort of value is involved. But this serves only as a comment, and has no other function.
- Under ***Data type*** you can select whether the value of the command is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.
- Under **Data input for command value** you can specify whether the command element is to have an orange data input on its left for the value to be passed. With all command elements, the value can either be entered directly in the command element or read in through a data input on the left side of the command element. In this way a motor, for example, can be controlled in a servo loop with a variable speed.

8.5.1 = (Assignment)

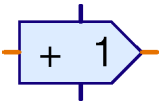


The **=** command assigns a value to the receiver. As a rule, it is used to assign a value to variables, timer variables, list elements or panel outputs.

But the **=** command is sent not only by command elements, but by all program elements with data outputs. All elements send **=** commands when the value of an output is altered. A Digital input element, for example, sends an =1 command when a sensor on the input is closed and an =0 command when the sensor is opened. But no command element is used to do this. Program elements with data outputs have, so to speak, **=** command elements built in.

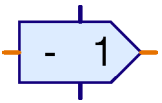
All ROBO Pro program element data inputs can process at least the **=** command. This makes the **=** command the most frequently used command in ROBO Pro.

8.5.2 + (Plus)



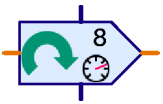
The **+** command is send to a variable or a timer variable to increase the value of the variable. Any desired value can be passed with the **+** command, and will be added to the variable. As the value passed with the command can also be negative, the value of the variable can also be decreased by this command. See Section 8.4.1 *Variable* on page 89 and Section 8.4.4 *Timer variable* on page 91.

8.5.3 – (Minus)



The **–** command is used similarly to the **+**command described above. The only difference is that value passed with the command is subtracted from the value of the variable.

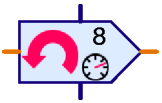
8.5.4 Right



The **Right** command is sent to motor output elements to switch on the element with clockwise rotation. See Section 8.7.4 *Motor output* on page 102.

The value is a speed from 1 to 8.

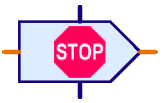
8.5.5 Left



The **Left** command is sent to motor output elements to switch the motor on in a counterclockwise direction. See Section 8.7.4 *Motor output* on page 102.

The value is a speed from 1 to 8.

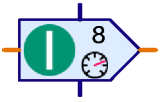
8.5.6 Stop



The **Stop** command is sent to a motor output element to stop the motor. See Section 8.7.4 *Motor output* on page 102.

No value is passed with the **Stop** command.

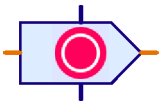
8.5.7 On



The **On** command is sent to a lamp output element to switch the lamp on. See Section 8.7.5 *Lamp output* on page 103. An **On** command can also be sent to a motor output element; it corresponds to the **Right** command. For motors, however, it is better to use the **Right** command, as the direction of rotation is then directly recognizable.

The value is the brightness or intensity, from 1 to 8.

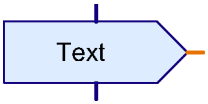
8.5.8 Off



The **Off** command is sent to a lamp output element to switch the lamp off. See Section 8.7.5 *Lamp output* on page 103. An **Off** command can also be sent to a motor output element; it corresponds to the **Stop** command.

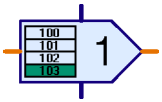
No value is passed with the **Off** command.

8.5.9 Text



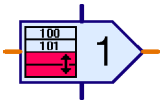
The **Text** command is a special command in that it doesn't send a command with a number, but rather a text of your choice, to the connected element. However, there is only one program element that can process the Text command, and that is a text display in a panel. You will find further information in Section 9.1.2 *Text display* on page 117.

8.5.10 Append value



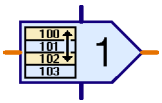
The **Append** command is a special command for list elements. See Section 8.4.5 *List* on page 92. The command is accompanied by a value, which is appended to the end of the list. If the list is already full, the command is ignored.

8.5.11 Delete value(s)



The **Delete** command is a special command for list elements. See Section 8.4.5 *List* on page 92. With this command, any number of elements can be deleted from the end of the list. The desired number is passed with the command as a value. If the value passed is greater than the number of elements in the list, all the elements in the list are deleted. In order to delete a list entirely, a **Delete** command with the maximum possible value of 32767 can be sent.

8.5.12 Exchange values

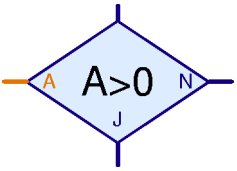


The **Exchange** command is a special command for list elements. See Section 8.4.5 *List* on page 92. With this command, any element of a list can be exchanged with the first element. The number of the element to be exchanged with the first element is passed with the command as a value. ***Important:*** the first element of a list has the number 0. If the value passed is not a valid element number, the list element will ignore the command.

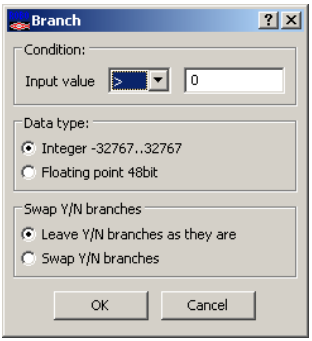
8.6 Compare, wait for, ... (Level 3)

The program elements in this group all serve for branching of program control or for delaying the continued running of the program.

8.6.1 Branch (with data input)



This program branch has an orange data input **A** on the left of the element. Via this, a value is read in, which often comes from an input element (see Section 8.7.1 to 8.7.6 from page 100). The data input **A** can also be linked to data outputs from variables, timer variables or operators (see Section 8.8 *Operators* on page 106. The element compares the value at the data input **A** with a fixed, but freely definable value. According to whether the comparison holds or not, the element branches to the **Y** or to the **N** exit.

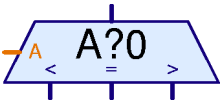


Properties window for the Branch

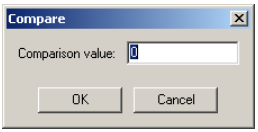
- Under **Condition** in the right-hand field you enter a value which is to be compared with the input value **A**. The usual comparison operators are available for the comparison.
- Under ***Data type*** you can select whether the received value is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130.
- If you select **Interchange Y/N connections**, the **Y** and **N** exits are exchanged as soon as you close the Properties window with OK. To return the Y/N connections to their initial positions, you can exchange them again.

The most commonly used comparison is **A>0**. That means that control branches to the **Y** exit if the value present at data input **A** is greater than 0. For example, digital inputs, which deliver a 1 or 0 value, can be evaluated in this way. But also timer variables and many other values can meaningfully be evaluated with the comparison **A>0**.

8.6.2 Comparison with fixed value



With the program element **Comparison with fixed value**, the value in the data input **A** can be compared with a fixed, but freely definable value. According to the value present at the data input **A** is greater than, less than, or equal to the fixed value, the comparison element branches to the right, left or middle exit. As a rule, the output of a variable or a list is connected to the data input **A**. The Compare element may be replaced by two Branch elements. In many cases, however, it makes for greater understandability if only one element is needed.



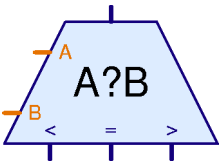
Note: This element does not exist for floating point numbers since floating point numbers are prone to rounding errors. Therefore it might not be reasonable to ask whether two values are

exactly the same. You can do a two-way comparison with a program branch. See section 8.6.1 *Branch (with data input)* on page 97.

Properties window for Compare

- Under **Comparison value** you can enter the constant value with which the value at input **A** is to be compared.

8.6.3 Compare

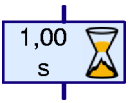


With the **Compare** program element, the two values at the data inputs **A** and **B** may be compared with one another. Depending on whether **A** is less than **B**, **A** is greater than **B**, or **A** equals **B**, the element branches to the left, right, or middle exit. The most common application for this is the comparison of a nominal value with an actual value. According to where the nominal value lies in relation to the actual value, then, for example, a motor can turn left or right or be stopped.

The **Compare** program element has no options to be set, and therefore no Properties window.

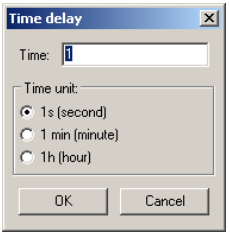
Note: This element does not exist for floating point numbers since floating point numbers are prone to rounding errors. Therefore it might not be reasonable to ask whether two values are exactly the same. You can do a two-way comparison with a comparative operator (see section 8.8.2 *Comparative operators (relational operators)* on page 106 and a program branch (see section 8.6.1 *Branch (with data input)* on page 97).

8.6.4 Time delay



With this element, a **Time delay** can be programmed into a procedure. The time delay starts when the element has its turn to be executed. As soon as the entered time delay is expired, the program continues running. See also Section 3.6.1 *Time delay* on page 22.

Properties window for Time delay:



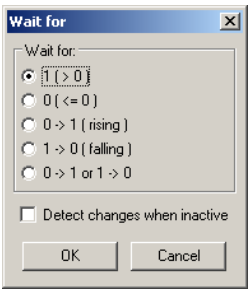
- Under **Time** you can enter the time delay. You can even use decimal fractions like 1.23.
- Under **Time unit** you can select seconds, minutes or hours as the unit of time. The time unit has, unlike the case for timer variables, no influence on the accuracy of the time delay. A time delay of 60 seconds and a time delay of 1 minute behave in exactly the same way.

In Expert mode (Level 5) an expanded Properties window is displayed, which is more like the Properties window for timer variables.

8.6.5 Wait for...

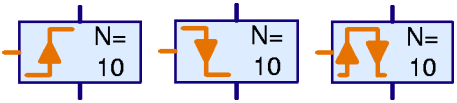
The **Wait for...** program element holds up program execution until a change has taken place or a specific state has been reached in the data input of the element. The element comes in five varieties: The element on the left waits until the value in the input has increased. For this purpose, not only changes from 0 to 1, but any increase, say for example from 2 to 3, will count. The second element waits until the value in the input has decreased, and the element in the middle waits for any change, regardless of direction. The third element is often used for pulse wheels. The fourth and fifth elements wait, not for a change, but for the state Yes (>0) or No (<=0) in the input. If the relevant state is already present, the element doesn't wait. The first three elements, on the other hand, always wait until a change is detected in the input.

Properties window for Wait for change



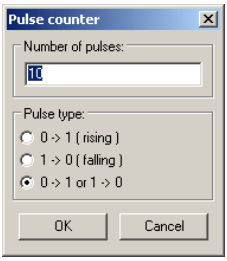
- Under **Type of change** you can choose between the five functions described above.
- If the button **Detect changes when not active** is pressed, the element also detects changes that took place when the element was not due to be executed. In this case, the element saves the last known value. When the element is executed again, it continues program execution immediately if the value has changed in the right way in the interim. In this way, there is less probability of missing a change, because the program just happened to be doing something else.

8.6.6 Pulse counter



This program element waits for a definable number of pulses at the data input on the left side before it continues program execution. This is very practical for simple positioning tasks with pulse wheels. For more demanding positioning, e.g. with a variable value, subprograms with variables must be used.

Properties window for Pulse counter



- Under **Number of pulses** you enter the number of pulses to be waited for before program execution is continued.
- Under **Pulse type** you can select between the three types of pulses: **0-1, 1-0** or any change.

The possibility of recognizing changes when the element is not active, as can be done with the simple **Wait for ...**, is not available for this element.

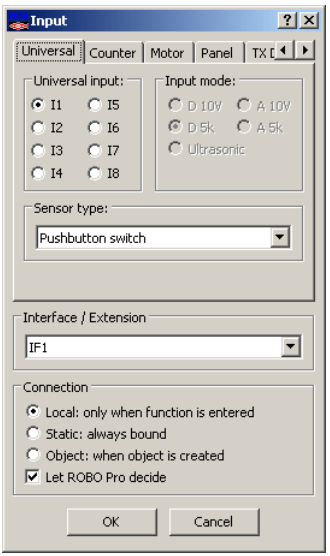
8.7 Interface inputs/outputs

This group of program elements contains all input and output elements. How to use these elements is explained in Chapter 4.4 *Level 3: Variables, panels & Co* on page 35.

8.7.1 Universal input

The TX Controller has 8 universal inputs **I1-I8** which can be used as digital or analog inputs. You can connect buttons and any sensor from the current fischertechnik line of products to these inputs.

Properties window for universal inputs:



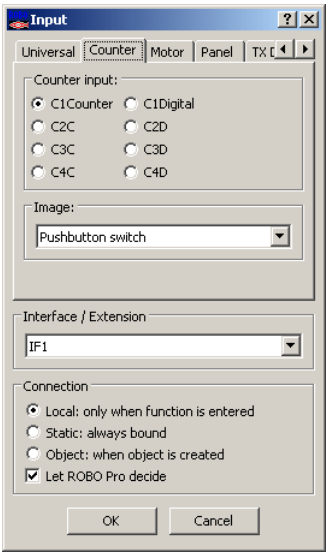
- Under **Universal input** you may select which of the Interface's inputs is to be used. Extension module inputs are selected under **Interface / Extension**.
- Under **Sensor type** you can select the sensor connected to the input.
- Under **Input mode** you select whether the input is an analog or digital input and whether it reacts to voltage or restistance or whether it is connected to an ultrasound distance sensor. You will learn more in section 11.5 *Universal inputs, sensor type and input mode* on page 125. ROBOPro automatically determines the input mode depending on the connected sensor. In Level 4 and above you can also select the imput mode independently. For example, this makes sense in the case of a phototransistor. For phototransistors ROBOPro sets the input mode to digital 5kOhm (D 5k). This way, you can use the phototransistor in conjunction with a lens lamp as a light barrier that is either interrupted (= 0) or closed (= 1). However, if you select the input mode analog 5kOhm (A 5k) for the photoresistor, you can differentiate between many shades of light and dark.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Connection** you can select whether the input is always connected or only when the subprogram containing the input is run. This only makes a difference if other global elements such as global variables and operators are connected to the input in a subprogram.

On closer examination, there is only one type of program element for all types of inputs. You can switch input times at any time via the tabs at the top of the Properties window. This is particularly useful for switching between switch, IR and panel inputs.

8.7.2 Counter input

In addition to the 8 universal inputs **I1-I8** the TX Controller offers 4 counter inputs **C1-C4**. You can only connect digital sensors and the encoders of encoder motors to conter inputs. For every counter input such as input **C1** there are two different counter inputs such as **C1C** and **C1D** in ROBO Pro. The input C1D behaves like a regular digital input. However, the input C1C counts the number of pulses received by input C1. You can reset the counter by sending a reset command to the respective motor element. The counters are used to control encoder motors and must be used for other purposes only if no encoder motor is connected to the respective motor output (for example, M1 for C1).

Properties window for Counter inputs:



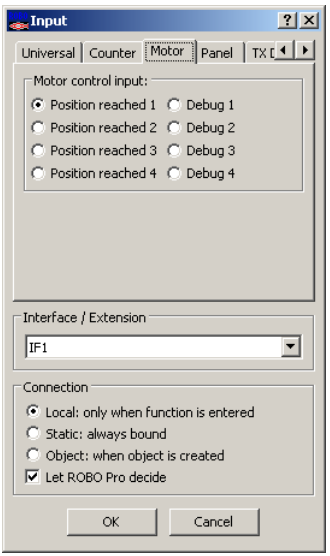
- Under **Counter input** you can select the desired counter or digital input.
- Under **Image** you can select an image of the sensor connected to the input.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Connection** you can select whether the input is always connected or only when the subprogram containing the input is run. This only makes a difference if other global elements such as global variables and operators are connected to the input in a subprogram.

It again becomes clear on the Properties window for counter inputs that for all inputs ROBO Pro uses a single element, which can be switched between input types through the tabs. For simplicity, however, separate input elements are available ready for selection in the element window.

8.7.3 Motor position reached

These inputs are not real inputs but logical inputs to control encoder motors. You will learn more in section 11.6.2 *Extended Motor Control in Level 3* on page 126.

Property window for motor inputs:



- Under **Motor control input** you can select which motor you would like to query the position reached signal for.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Connection** you can select whether the input is always connected or only when the subprogram containing the input is run. This only makes a difference if other global elements such as global variables and operators are connected to the input in a subprogram.

8.7.4 Motor output

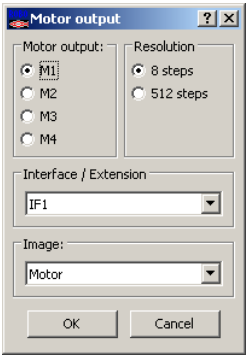
The **Motor output** element allows one of the 4 two-pole motor outputs of a ROBO Interface or an Intelligent Interface to be controlled. A motor output always uses two Interface connections, whereas a lamp output only uses one connection. You can find out more about the difference between motor and lamp outputs in Sections 8.1.6 *Motor output* on page 77 and 8.1.8 *Lamp output*.

A command must be sent via a command element to a motor output in order to switch the output. A motor element can process the following commands:

Command	Value	Action
Right	1 to 8	The motor turns clockwise with a speed of 1 to 8
Left	1 to 8	The motor turns counterclockwise with a speed of 1 to 8
Stop	none	The motor stops
On	1 to 8	As for Right
Off	none	As for Stop
=	-8 to 8	Value -1 to –8: the motor turns counterclockwise Value 1 to 8: the motor turns clockwise Value 0: the motor stops

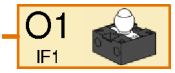
In addition, a motor element can receive commands for extended motor control (synchronous, distance, reset), as explained in section 11.6.2 *Extended Motor Control in Level 3* on page 126.

Properties window for Motor elements;



- Under **Motor output** you can select which of the Interface's output connections are to be used. You can select extension module outputs under **Interface / Extension**.
- Under **Resolution** you can select whether you would like to control the intensity of the output in 8 steps (1-8) or in 512 steps (1-512).
- Under **Interface / Extension** you can select whether you want to use an output of the Interface or an output of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Image** you can select an image of the load connected to the output. In most cases this will be a **motor**. But you can also connect an **electromagnet**, a **solenoid valve** or a **lamp** to a motor output.

8.7.5 Lamp output

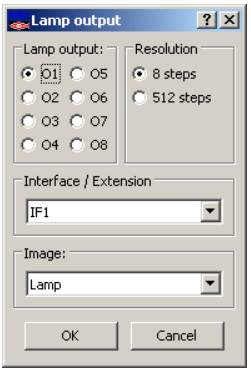


The **Lamp output** element allows one of the 8 single-pole lamp outputs O1-O8 of a ROBO Interface or an Intelligent Interface to be controlled. A lamp output only ever uses one output connection of the Interface. The other connection of the load device is connected to the ground socket. You can only switch on or off a load device connected in this way; you can't reverse its polarity. You can learn more about the difference in Sections 8.1.6 *Motor output* on page 77 and 8.1.8 *Lamp output (Level 2)*.

A command that switches the output must be sent via a command element to a lamp element. A Lamp element can process the following commands:

Command	Value	Action
On	1 to 8	The lamp is switched on with a brightness of 1 to 8
Off	none	The lamp is switched off
=	0 to 8	Value 1 to 8: the lamp is switched on Value 0: the lamp is switched off

Properties window for lamp output elements:



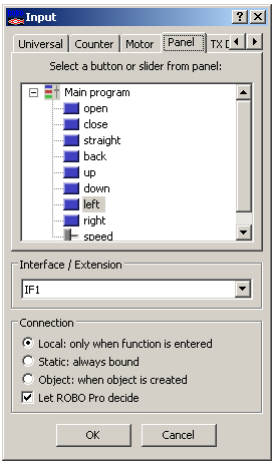
- Under **Lamp output** you can select which of the Interface's output connections is to be used. You can select extension module inputs under **Interface / Extension**.
- Under **Resolution** you can select whether you would like to control the intensity of the output in 8 steps (1-8) or in 512 steps (1-512).
- Under **Interface / Extension** you can select whether you want to use an output of the Interface or an output of an extension module or of another Interface. You will learn more about this in Chapter 7 *Controlling several Interfaces* on page 70.
- Under **Image** you can select an image of the load device connected to the output. In most case this will be a **lamp**. But you can also connect an **electromagnet**, a **solenoid valve** or even a **motor** to a lamp output. But a motor connected to a lamp output can only ever rotate in one direction.

8.7.6 Panel input

ROBO Pro offers you the possibility of designing your own panels for your models. You can learn more about this in Chapter 8.9 *Panel elements and panels: overview* on page 110. This makes it convenient for you to control your models from the computer. Push buttons, slider controls and data entry elements are available for use in a panel. The state of these elements can be queried in the program via the **Panel input** element. Push buttons return a value of 0 or 1. Slider controls return a value in a user-definable range (by default, 0 to 100).

Panels can only be used in online mode. You can find out more about this in Section 3.7 *Online and download operation-what's the difference?* on page 24.

Properties window for Panel inputs:



One panel is associated with each main program or subprogram. The panel elements are listed under the name of the respective program. If you have not yet defined any panel elements, then no elements will appear in the list. So you must first design the panel before you can link a panel input with a panel element.

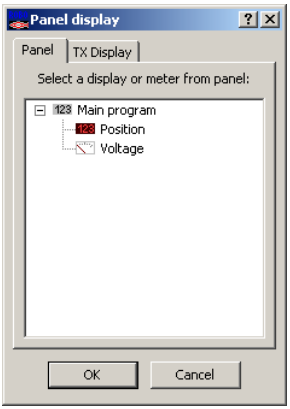
The selection under **Interface / Extension** is ignored in the case of panel input, as we are not dealing here with actual inputs on an Interface module.

8.7.7 Panel Output

ROBO Pro offers you the possibility of designing your own panels for your models. You can learn more about this in Chapter 8.9 *Panel elements and panels: overview* on page 110. Alongside push buttons and other input elements to control your model, you can also insert display elements in your panel. In these display elements you can display, for example, the axis coordinates of a robot or the state of an end switch. You alter the value to be displayed by inserting a **Panel output** element in your program and sending the element an **=** command, e.g. by connecting a variable, an analog input or a command element to it.

Panels can only be used in online mode. You will learn more about this in Section 3.7 *Online and download operation-what's the difference?* on page 24.

Properties window for Panel displays:

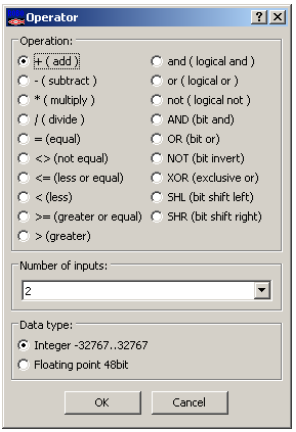


One panel belongs to every main program or subprogram. Panel displays are listed under the name of the respective programs. If you have not yet established any panel elements, then no elements will appear in the list. So you must first draw the panel before you can link a panel input to a panel element.

8.8 Operators

All program elements in this group are what are called operators. Operators have one or more orange data inputs. The values from the data inputs are combined by the operator to create a new value which is transmitted from the operator's output by means of an = command.

Properties window for operators

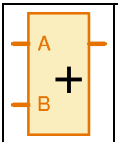
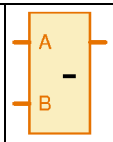
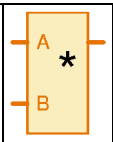
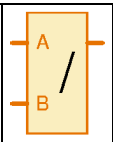
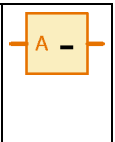


All operators use the same Properties window. Through the Properties window, you can even transform an operator into a different operator.

- Under **Operation** you can set how the operator is to combine its inputs. The individual functions are explained in the next two Sections.
- Under **Number of inputs** you can set the number of inputs the operator is to have.

8.8.1 Arithmetic operators

ROBO Pro makes the four basic operations of arithmetic available to you as operators. With two inputs, the symbols look like this:

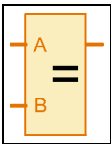
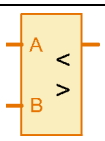
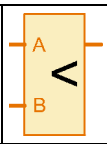
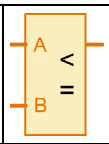
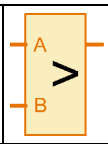
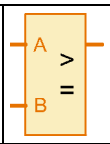
				
Plus	Minus	Times	Divided by	Minus
A + B	A - B	A * B	A / B	- A

If the **Minus** operator has more than two inputs, all subsequent input values are subtracted from the value in input A. If the Minus operator only has one input, the operator changes the sign of the input value.

If the **Divided by** operator has more than two inputs, the value in input A is divided by all further input values.

8.8.2 Comparative operators (relational operators)

There are 6 comparative operators to compare values:

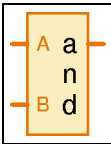
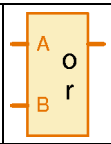
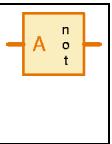
					
equal	not equal	less than	less than or equal to	greater than	greater than or equal to
A = B	A ≠ B	A < B	A ≤ B	A > B	A ≥ B

If a comparison is true, the output value is 1, otherwise 0. The output value is always a whole number even if the input values are floating point numbers.

Besides the not equal operator you can use any of the comparative operators with more than two inputs. The result then becomes 1 if the condition is true for A and B as well as for B and C, and so on. For example, this way you can determine with one single operator, whether a value lies within the given upper and lower bound.

8.8.3 Logical operators

ROBO Pro has three logical operators, which can be used for instance to combine digital inputs.

		
And	Or	Not
A>0 and B>0	A>0 or B>0	A <= 0

The logical operators interpret a value greater than zero as **yes** or **true** and a value less than or equal to zero as **no** or **false**. Digital inputs return a value of 0 or 1, so that 0 is interpreted as **false** and 1 as **true**.

The **And** operator sends an = command with the value 1 to the elements connected to its output if all inputs have the value true, i.e. a value >0. Otherwise the element sends an **=** command with the value 0.

The **Or** operator sends an **=** command with the value 1 to the elements connected to its output if at least one input has the value true, i.e. a value >0. Otherwise the element sends an **=** command with the value 0.

The **Not** operator sends an **=** command with the value 1 to the elements connected to its output if its input has the value false, i.e. a value <=0. Otherwise the element sends an **=** command with the value 0.

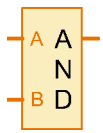
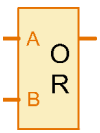
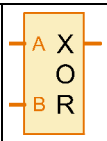
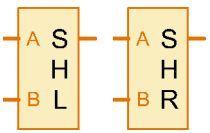
The function of logical operators can also be emulated with several Branch elements. But it often makes for much easier understanding to combine several inputs using operators.

8.8.4 Bit operators

A whole number variable in ROBOPro consists of 16 single bits. Any one of these bits can store a 0 or a 1. These bits become a number by assigning each one of them a power of two:

Bit	Numerical value	Bit	Numerical value
0	$1 = 2^0$	8	$256 = 2^8$
1	$2 = 2^1$	9	$512 = 2^9$
2	$4 = 2^2$	10	$1024 = 2^{10}$
3	$8 = 2^3$	11	$2048 = 2^{11}$
4	$16 = 2^4$	12	$4096 = 2^{12}$
5	$32 = 2^5$	13	$8192 = 2^{13}$
6	$64 = 2^6$	14	$16384 = 2^{14}$
7	$128 = 2^7$	15	$-32768 = 2^{15}$

For example, for the number 3 the bits 0 and 1 are set to 1 because $2^0 + 2^1 = 3$ ist. Bit operators carry out the same operations as logical operators with the exception that they do it for every individual bit. Thus 3 AND 6 yields the value 2 because the bit 2^1 is the only one that is set in both $3 = 2^0 + 2^1$ and in $6 = 2^1 + 2^2$. Please note that the numerical value 32768, for which only the bit 2^{15} is set to 1, has a special meaning in ROBOPro and is used for error or blank. To generate a variable with this value you simply enter nothing (blank) for its value.

				
And	Or	Not	Exclusive Or	Shift left/right
Bit is set if it is set in A and B	Bit is set if it is set in A or B	Bit is set if it is not set in A	Bit is set if it is set in A and B and not in both	The bits in A are shifted B places to the left (toward higher bits) or to the right (toward lower bits).

8.8.5 Functions

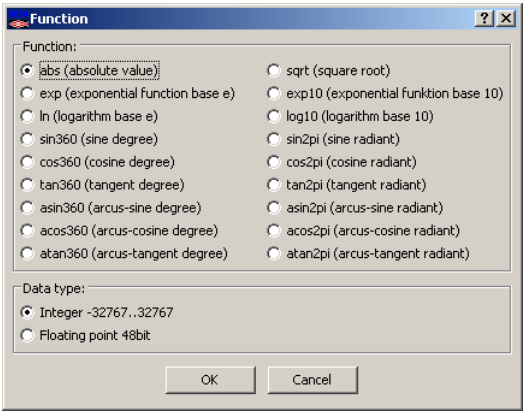
Functions are similar to operators but they always have only one input. Functions include trigonometric functions, roots, exponential and logarithmic functions.

Note 1: In many cases, functions are difficult to compute. Since the TX Controller makes sure that every process can carry out a command at least 1000 times per second, the number of functions that can be evaluated in one command is limited. Networks of organge data lines are always processed in one command and are not split up. Therefore one should not call too many functions in a row in an orange network.

Note 2: ROBOPro does not use arithmetics with extended precision to compute functions. Therefore, the precision of results is typically about 2 bits less than the maximal possible precision of the 48-bit floating point format. The precision of the results is estimated by ROBOPro and saved in the result.

Property window for functions

All functions use the same property window.



- Under **Function** you can select which mathematical function the element will compute. The single functions are explained in the following two sections.
- Under ***Data type*** you can select whether the result of the function is a whole number or a floating point number. Also see 12 *Working with decimals* on page 130. Except for the **abs** function, all functions are available as floating point only.

Basic functions

abs	**Absolute value:** Returns the positive value of the input, for example 3.2 for -3.2
sqrt	**Square root**: Returns the square root of the input, for example 1.4142… for 2.0

Exponential and logarithmic functions

exp	**Exponential function base e:** Returns for an input x the x-th power of Euler's number e, that is e^x
exp10	**Exponential function base 10:** Returns for an input x the x-th power of 10, that is 10^x or 100.0 for x=2.0
log	**Logarithm base e:** Returns for an input x the power Euler's number must be raised to to obtain x.
log10	**Logarithm basis 10:** Returns for an input x the power 10 must be raised to to obtain x. For example, for x=1000 we get the result 3.0

Trigonometric functions and inverse functions

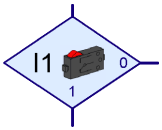
All trigonometric functions and inverse functions exist for two different angle measures, that is for degrees (1 full circle = 360 degrees) and radians (1 full circle = 2 pi).

sin360 / sin2pi	**Sine:** Returns for an input x the sine of the angle x
cos360 / cos2pi	**Cosine:** Returns for an input x the cosine of the angle x
tan360 / tan2pi	**Tangent:** Returns for an input x the tangent of the angle x
asin360 / asin2pi	**Arcsine:** Returns for a sine value x the matching angle
acos360 / acos2pi	**Arccosine:** Returns for a cosine value x the matching angle

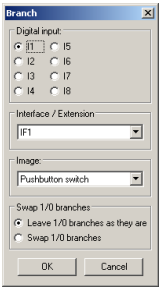
atan360 / atan2pi	**Arctangent:** Returns for a tangent value x the matching angle

8.9 ROBO Interface

8.9.1 Digital Branch (ROBO Interface)



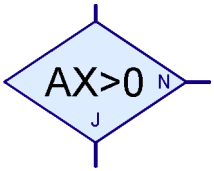
With this Branch you can direct program control, according to the state of one of the digital inputs **I1** to **I8**, in one of two directions. If, for example, a sensor on the digital input is closed (=1), the program branches to the **1** exit. On the other hand, if the input is open (=0), the program branches to the **0** exit.



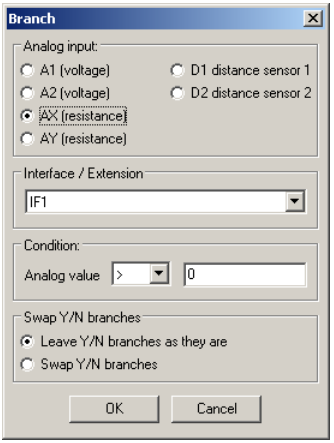
If you right-click on the element, the Properties window is displayed:

- Buttons **I1** to **I8** allow you to enter which of the Interface's inputs is to be queried.
- Under **Interface / Extension** you can select whether you want to use an input of the Interface or an input of an extension module or of another Interface. You can find out more about this in Chapter7 *Extension modules and controlling several Interfaces* on page 70.
- Under **Image** you can select an image for the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed contacts.
- Under **Interchange 1/0 connections** you can interchange the positions of the 1 and 0 exits of the Branch. Normally the 1 exit is below and the 0 exit is on the right. But often it's more practical to have the 1 exit on the right. Press **Interchange 1/0 connections** and then the two connections will be interchanged as soon as you close the window with **OK**.

8.9.2 Analog Branch (ROBO Interface)



As well as the digital inputs, the ROBO Interface has 6 Analog inputs: 2 resistance inputs AX and AY, two voltage inputs A1 and A1, as well as two inputs for distances sensors D1 and D2. With this Branch you can compare the value of an analog input with a fixed number and, according to the result of the comparison, branch to the Yes (Y) or No (N) exit.

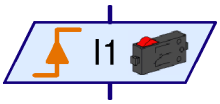


If you right-click on the element, the Properties window is displayed:

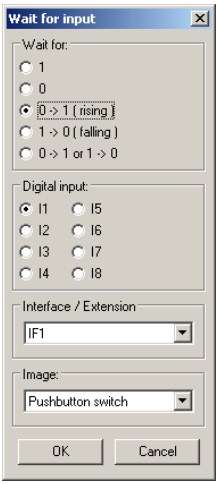
- Under **Analog input**, you can select which of the Interface's inputs is to be queried. All analog inputs return a value between 0 and 1023. You can find further information about the various analog inputs in Section 8.7.2 *Analog input* on page 101.

- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You can find further information about the various analog inputs in Chapter 7 *Extension modules and controlling several Interfaces* on page 70.
- Under **Condition** you can select a comparison operator such as less than (<) or greater than (>) and enter the comparison value. The comparison value should lie in the range from 0 to 1023. When you start a program containing a Branch for analog inputs in online mode, the current analog value is displayed.
- Under **Interchange Y/N connections** you can exchange the position of the Y and N exits of the Branch. Normally the Yes (Y) exit is below and the No (N) exit is on the right. But often it's more practical to have the Yes exit on the right. Press **Interchange Y/N connections** and the Y and N connections are swapped as soon as you close the window with **OK**.

8.9.3 Wait for input (ROBO Interface)



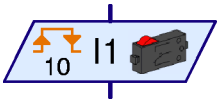
The **Wait for Input** element waits until one of the Interface's inputs is in a particular state or until it changes in a particular way.



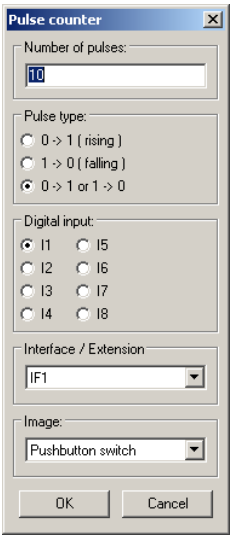
If you right-click on the element, the Properties window is displayed:

- Under **Wait for** you can select the type of change or the state to be waited for. If you select **1** or **0**, the element waits until the input is closed (1) or open (0). If you choose **0 -> 1** or **1 -> 0**, the element waits until the state of the input ***changes*** from open to closed (0->1) or from closed to open (1->0). In the last case, the element waits until the state of the input changes, regardless of whether it's from open to closed or vice versa. To help you understand this further, it is explained in Section 3.6 *Other program elements* on page 22 how you can emulate this element with the Branch element.
- Under **Digital input** you may enter which of the Interface's inputs **I1** to **I8** is to be queried.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You can find out more about this in Chapter 7 *Extension modules and controlling several Interfaces* on page 70.
- Under **Image** you can select an image for the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed contacts.

8.9.4 Pulse counter (ROBO Interface)



Many fischertechnik model robots also use pulse wheels. These gear wheels operate a sensor four times for every revolution. With these pulse wheels you can turn a motor on for a precisely defined number of revolutions rather than for a given time. To do this, you need to count the number of pulses at an input of the Interface. For this purpose there is the **Pulse counter** element, which waits for a user-definable number of pulses.



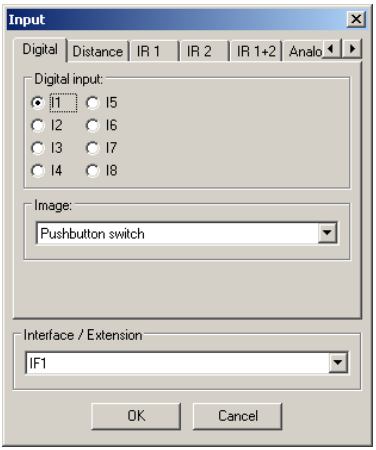
If you right-click on the element, the Properties window is displayed:

- Under **Pulse type** you can select the type of pulse to be counted. If you choose **0 -> 1** (rising), the element waits until the state of the input has changed from open to closed (0->1) the number of times you have specified under **Number of pulses**. If you choose 1 -> 0 (falling), the element waits until the state of the input changes from closed to open (1->0) the specified number of times. With pulse wheels, however, the third possibility is used more often. Here the element counts both **0 -> 1** and **1 -> 0** changes, so that 8 pulses are counted per revolution of a pulse wheel.
- Under **Digital input** you may enter which of the Interface's 8 inputs **I1** to **I8** is to be queried.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Extension modules and controlling several Interfaces* on page 70
- Under **Image** you can select an image for the sensor connected to the input. Digital inputs are mostly used with push-button switches, but often also with phototransistors or reed contacts.

8.9.5 Digital input (ROBO Interface)

The value of one of the digital inputs I1 to I8 can be queried via the **Digital input** element. If the two sockets belonging to the input on the Interface are electrically connected, the digital input element returns a value of 1 on its orange connection, otherwise a value of 0.

Properties window for Digital inputs:



- Under **Digital input** you may select which of the Interface's inputs is to be used. Extension module inputs are selected under **Interface / Extension**.
- Under **Image** you can select an image of the sensor connected to the input. In most cases this will be a **mini-push-button** sensor. A **reed contact** is a switch that reacts to magnetic fields. Even a **phototransistor** can be connected to a digital contact, although it is really an analog sensor. You can use a lamp with lens together with the phototransistor connected to a digital input as a photoelectric beam, which is either interrupted (=0) or closed (=1). On the other hand, if you connect the phototransistor to an *Analog input*, you can distinguish many gradations between light and dark.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Extension modules and controlling several Interfaces* on page 70.

On closer examination, there is only one type of program element for all types of inputs. You can switch input times at any time via the tabs at the top of the Properties window. This is particularly useful for switching between switch, IR and panel inputs.

8.9.6 Analog input (ROBO Interface)

The value of one of the analog inputs can be queried via the **Analog input** element. Unlike digital inputs, which can only return a value of 0 or 1, analog inputs can distinguish fine gradations. All analog inputs return a value between 0 and 1023. The ROBO Interface, however, has various kinds of analog inputs, which measure various physical quantities. There are analog inputs for resistance measurements, for voltage measurements and for a special distance-measuring sensor.

Input	Input type	Measurement range
A1, A2	Voltage inputs	0-10,23V
AX, AY	Resistance inputs	0-5,5kΩ
D1, D2	Distance sensor input	ca. 0-50cm
AV	Power supply voltage	0-10V

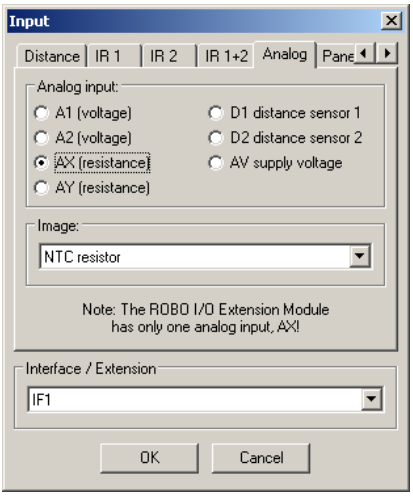
The usual fischertechnik sensors, NTC resistor, phototransistor and photoresistor, transform the quantity measured (temperature of light intensity) into a resistance. Therefore, you must connect these sensors to the **AX** or **AY** input. The voltage inputs **A1** and **A2** are designed for all sensors that produce a voltage between 0 and 10V.

There is no socket on the ROBO Interface for the **AV** input. It is always linked to the Interface's supply voltage. In this way you can, for example, monitor the battery voltage and put the model into its exit position before the battery is flat.

The distance sensor inputs **D1** and **D2** accept connection to special fischertechnik sensors that can measure the distance to, for example, an obstacle.

The Intelligent Interface has only two analog inputs, EX and EY. These correspond to the AX and AY inputs of the ROBO Interface. The other analog inputs cannot be used with the Intelligent Interface!

Properties window for Analog inputs:



- Under **Analog input** you can select the desired analog input using the table above.
- Under **Image** you can select an image of the sensor connected to the input.
- Under **Interface / Extension** you can select whether you wish to use an input of the Interface or an input of an extension module or of another Interface. You will learn more about this in Chapter 7 *Extension modules and controlling several Interfaces* on page 70.

It again becomes clear on the Properties window for analog inputs that for all inputs ROBO Pro uses a single element, which can be switched between input types through the tabs. For simplicity, however, separate input elements are available ready for selection in the element window.

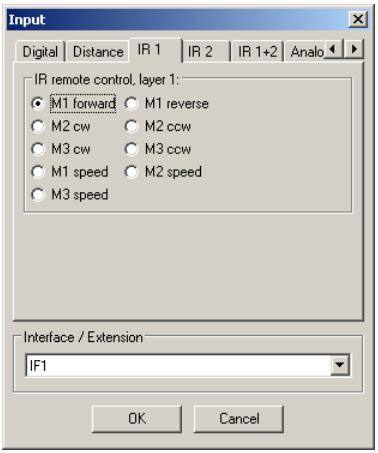
8.9.7 IR Input (ROBO Interface)

The ROBO Pro Interface has a built-in infrared receiver for the hand-held transmitter from the fischertechnik ***IR Control Set***, item number 30344. The infrared hand transmitter is very useful, not only for remote control, but also generally as a keyboard for your models. There are two receivers for the ***IR Control Set***, and you can switch between them with the buttons **1** and **2** on the handset. So you can assign two functions in your ROBO Interface to each button of the handset. You can switch between two assignments with the shift keys **1** and **2**. Alternatively, the **1** and **2** keys can be used as ordinary keys.

In the Properties window of an IR input you can use the tab bar at the top to switch between **IR 1**, **IR 2** and **IR 1+2**. If you have selected **IR 1**, the IR input element only returns a 1 if the corresponding key on the transmitter is depressed and the transmitter has previously been set via the **1** key to assignment 1. If you select **IR 2**, on the other hand, the transmitter must have been set to assignment 2 using the **2** key.

But if you select **IR 1+2**, the setting of the handset doesn't matter. In this case, you can also use the 1))) and 2))) keys as inputs.

In the program element this choice is displayed by means of a white 1 or 2 in the lower right of the handset symbol In the case of **IR 1+2** no number is displayed in the program element.



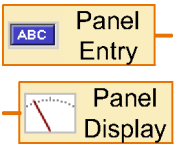
9 Panel elements and panels: overview

In ROBO Pro you can define your own panels. Panels make it less cumbersome to control complex models. A panel is displayed on your PC screen. Panels only work in online mode. On this subject, see Section 3.7 *Online and download operation-what's the difference?_Debug-__Online-_oder Download-Betri* on page 24.

To create a panel, you select **Panel** in the function bar.

In the empty gray field below you can then insert panel elements. A panel always belongs to the main program or subprogram in which you were when you created the panel. Therefore it is important that you always select the right subprogram in the subprogram bar before creating a panel. Panels are generally created under the **main program**.

Panels contain displays and control elements. With displays, you can display for example variable values or text messages. Control elements, on the other hand, function as additional sensors of analog inputs.



To every panel element that you insert in the panel there is a corresponding element in the program: a **Panel input** (for control elements) or a **Panel output** (for displays). You establish the link between your program and your panel through these program elements. You find them in the **Inputs, outputs** element group. A different symbol is displayed according to which type of panel element you link to these program elements. But in the element list there are only two elements: one for displays and one for control elements.

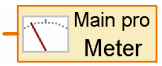
9.1 Displays

Displays are used in a similar way to Interface outputs. You can set the value of a display with an **=** command.

9.1.1 Meter

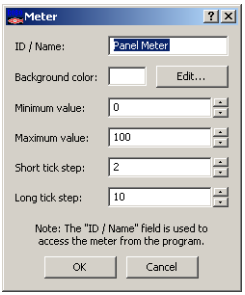
The **Meter** is based on an analog instrument with pointer. It is mostly used to display the value of analog inputs; but you can also use it for variables or other program elements.

The meter is controlled from the program via a panel output. You will find the **Panel output** in the element group **Inputs, outputs.**



You set the value of the meter by sending an **=** command to the corresponding panel output in the program. Almost all program elements with data outputs send an **=** command when their value changes. You can connect analog inputs or variables for example directly to the panel output.

Properties window for meters

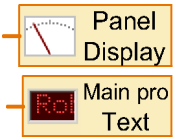


- Under **ID / Name** you should first enter a name for the meter. The name is important so that you can distinguish between more than one meter in your program.
- Under **Background color** you can set another color than white.
- Under **Minimum value** and **Maximum value** you specify the values corresponding to the needle positions at the left and right ends of the scale. If one of the values is less than 0 and the other greater than 0, a particularly long 0 stroke is drawn.
- The scale consists of long and short strokes. The distance between the long and short strokes is entered under **Step size short / long marks.** If both have the same value, only long marks are visible.

9.1.2 Text display

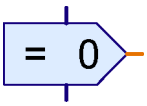
In a text display you can show numerical values, text, or a mixture of the two.

The text display is controlled from the program via a panel output. You will find the **Panel output** in the element group **Inputs, outputs**.

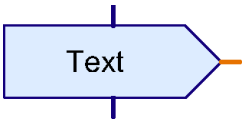


As soon as you have linked the Panel output with a text display by means of its Properties window, the symbol changes and the name of the panel (e.g. Main) and of the display (e.g. Text) appear.

There are two ways in which you can set the text in the display:



- You set the content of the display by sending an **=** command to the corresponding panel output in the program. This is very practical if you want to use the display to display the value of a variable or other program element, because most program elements automatically send an **=** command through their data output whenever their value changes. The **=** command overwrites only the last 6 characters of the display. You can fill the rest of the display with a pre-entered text. In this way you can supply some explanatory text for the value in the display. If it is a multi-line display, you can also put the explanatory text in a line of its own. In multi-line displays only the last 6 characters of the last line are overwritten by an **=** command.



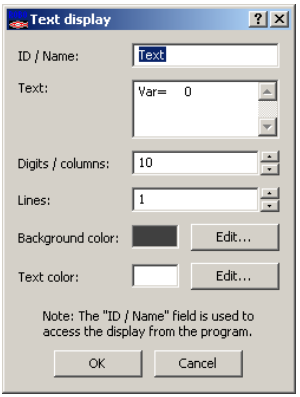
- With the Text command you can set the content of the display as you wish. The Text command is a special command element in that it can send, not just a number, but a whole text, through its output. Like an ordinary command element, the **Text** command element can also have a data input. In this case you can build the numerical value from the data input into the text. If you send a Display element multiple **Text** commands, the texts are concatenated. In this way you can mix numbers and text at will.

Control characters in Text commands

The following control characters can be used in the **Text** command element to achieve particular effects.

Control character	Effect
######	Outputs the value in the data input as a 5-digit number + sign character.
##.##	Outputs the value in the data input as a number with two decimal places, with a **period** as separator.
##,##	Outputs the value in the data input as a number with two decimal places, with a **comma** as separator.
\c	Clear display and insert subsequent text at the beginning of the display.

Properties window for text displays

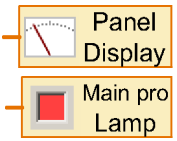


- Under **ID / Name** you should first enter a name for the display. The name is important so that you can distinguish between more than one display in your program.
- Under **Text** you enter the content of the display. This content is retained until you send a command to the display from the program. If you send an **=** command to the display, only the last 6 characters of the display contents are overwritten. The beginning of the text is retained so that you can display a note before the number saying what kind of number it is. In the example illustrated, the text **Var=** is retained. The display has 10 characters, and so 10-6=4 characters are retained.
- Under **Digits/columns** and under **Lines** you can set how many characters the display should allow room for. In a multi-line display you can display a note like **Var=** or **Visitors** in a line of its own.
- Under **Background color** and **Text color** you can alter the color design of the display. Click on **Edit ...** to select a color or to define your own color.

9.1.3 Display lamp

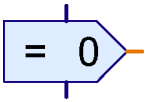
The **Display lamp** is the simplest type of display. It functions in a similar way to a fischertechnik lamp component connected to an Interface output.

The lamp is controlled from the program via a panel output. You will find the **Panel output** in the element group Inputs, outputs.

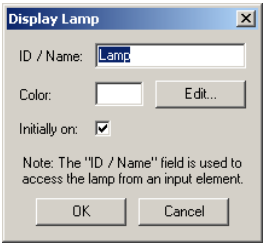


As soon as you have linked the Panel output with a display lamp by means of its Properties window, the symbol changes and the name of the panel (e.g. Main) and of the lamp appear.

You can switch the lamp on or off by sending the corresponding panel output an **On** or **Off** command, as you would also do for a real lamp output. You can also switch a display lamp on or off via an **=** command. If the value is greater than 0, the lamp is switched on If the value is less than or equal to 0, the lamp is switched off



Properties window for display lamps



- Under **ID / Name** you should first enter a name for the display lamp. The name is important so that you can distinguish between more than one display lamp in your program.
- Under **Color** you can change the color of the display lamp. To do this, click on the **Edit** button.
- If **Initially on** has a cross next to it, the display lamp is on until the corresponding program element receives a command. Otherwise the display lamp is initially off.

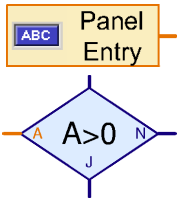
9.2 Control elements

Control elements are used in a similar way to Interface inputs.

9.2.1 Button

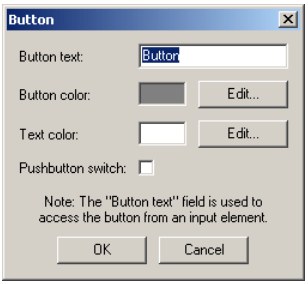
You can use the **Button** panel element like a fischertechnik sensor or switch connected to one of the inputs of the Interface.

The Button is queried from the program via a **panel input**. You will find the **Panel input** in the element group **Inputs, outputs**.



You can connect the panel output associated with the button, like an Interface digital input, to any program element with a data input, for example to the **Branch** element. If the button is depressed it returns 1 as its value, otherwise 0.

Properties window for buttons

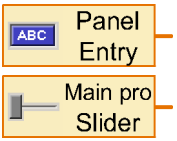


- Under **Inscription text** you can enter the inscription for the button. This is at the same time the name by which the button is accessed from the program. In the case of the button, there is no additional Name/ID field as found with the other panel elements.
- You can change the color design of the button under **Button color** and **Text color**. To do this, click on **Edit ....**
- If a check mark appears by **Pressure switch**, the button functions as a switch rather than a sensor. On the first click on the button it is pushed in, and then remains depressed until the second click. Otherwise, the button works like a sensor and springs open again straight away when it is released.

9.2.2 Slider

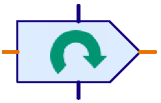
You can use the **Slider** like a potentiometer connected to an analog input of the Interface. Unlike the button the slider can return not only the values 0 and 1, but many different values, like an analog input. The range of values can be set through the Properties window. The slider can be used for example to set the motor speed between 1 and 8.

The Slider is queried from the program via a panel input. You will find the **Panel input** in the element group **Inputs, outputs**.

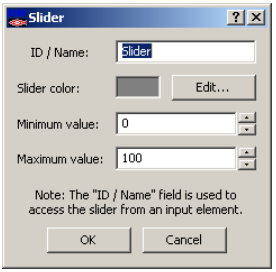


As soon as you have linked the Panel output with a slider by means of its Properties window, the symbol changes and the name of the panel (e.g. Main) and of the slider appear.

You can connect the panel output associated with the slider, like an Interface analog input, to any program element with a data input. Very often the slider is connected to a command element with a data input, so that the slider controls the speed of a motor.



Properties window for Sliders

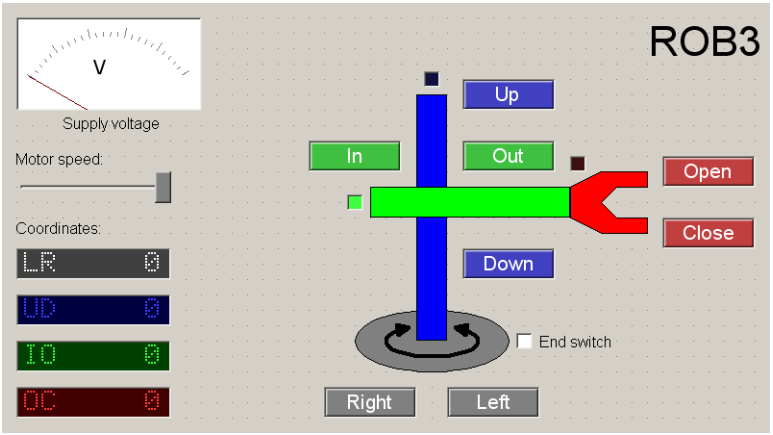


- Under **ID / Name** you should first enter a name for the slider. The name is important so that you can distinguish between more than one slider in your program.
- Under **Slider knob color** you can change the color of the slider knob. To do this, click on **Edit**.
- Under **Minimum value** and **Maximum value** you enter the value range for the slider. If you want to use the slider to control the speed of a motor, the value range should go from 1 to 8.

ROBO Pro has the usual drawing functions. You will them functions in the element group window under **Draw**. In the subgroup **Shapes** are contained drawing tools for various basic geometric shapes. In the **Text** subgroup you will find text-writing tools for various font sizes. The other sub-groups contain functions to alter color and line thickness.

10 Drawing functions

With drawing functions you can illustrate your panels and programs, to make their function clearer. Here for example is illustrated a user-designed panel for a robot.



The buttons, co-ordinate displays and end switch lamps are kept in each case in the same color as the respective individual axes in the schematic drawing of the robot. This results in panel that is very easy to understand.

The application of the drawing functions should present no great difficulties. So only a few points that might not be immediately clear are presented in the following:

- Graphical objects like rectangles and circles are **not** delineated as in many programs by holding down the mouse button, but through two mouse clicks, one in the upper left corner and one in the lower right corner.
- Text is not edited in a dialog window, but directly in the working area. When you insert a new text object, initially only a bright blue frame appears. You can now simply type at the keyboard and the text you type will appear directly in the working area. You can also insert text from the clipboard with CTRL+V.
- Once you have drawn an object, you can edit it by moving the small blue handles. There are also handles for turning and distorting objects. A rectangle has two handles at the upper left. If you displace the second, larger handle, you can round off the corners of the rectangle. You can exit editing mode by right-clicking with the mouse or by pressing the **ESC** key.
- If want to edit the object later, select the **Edit** function in the **Draw** menu. If you the click on an object, the bright blue handles will appear again.
- Many objects have two or more editing and drawing modes. While drawing or editing an object, you can switch between the individual modes with the TAB key on the keyboard. In the case of a circle, for example, you may select whether you would like to specify two boundary points or the center and one boundary point. In the case of polygons, you can change between point editing and functions like “rotate”. With text objects, you can switch between editing the text and changing the font size or angle of rotation.

- In the **Draw** menu there is functions to put the object **in the foreground / background**. With this function you can put all selected objects (drawn in red) forward or back, so that the obscure other objects or are obscured by them.
- With the **Raster snap** function in the **Draw** menu you can switch on or off the character matrix. You should however take note that the matrix is switched on when you are editing your program, as all program elements are aligned to the matrix.
- You can alter the alignment of text objects by pressing CTRL+ a key from 1-9 on the numeric keypad. But this only works if the Num-Lock light on the keyboard is on. If not, you must first press the NUM key.

11 New Functions for the ROBO TX Controller

ROBOPro 2.X can be used with the previous ROBO Interface as well as with the new ROBO TX Controller. You can develop ROBOPro programs such, that they run without changes on the previous interfaces as well as on the new interface. But since there are differences between the interfaces in the inputs and outputs, this is not true for any ROBOPro program. For example the ROBO TX Controller has 8 universal inputs, which can all be used as analog input for resistance values as well. The ROBO Interface, by contrast, has only two resistance inputs (AX and AY). On the other hand the ROBO Interface has an internal input for supply voltage. This can be measured by the ROBO TX Controller as well, using an universal input.

11.1 Installation of the ROBO TX Controller USB-driver

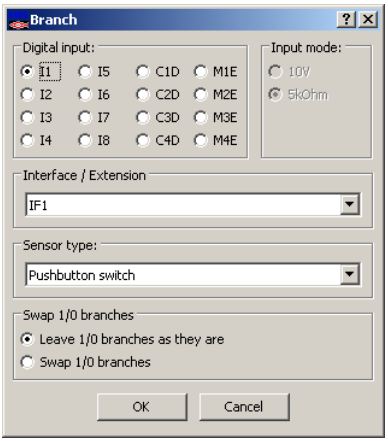
You can find the USB driver for the ROBO TX Controller in the ROBOPro installation folder in subfolder \USB-driver installation\TXController. There you can select the driver matching your Windows operating system. Except of the driver folder, the installation works in the same way as for the ROBO Interface (see also *Installing the USB driver for the ROBO Interface* on page 4).

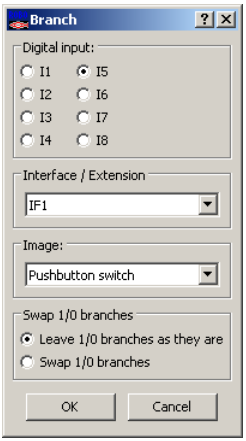
11.2 Environment (Level 1 and above)

In order to show only those options during the development of a program, which are actually supported by the target interface, you first select via the toolbar button, if a program is designed for the ROBO Interface or the ROBO TX Controller.

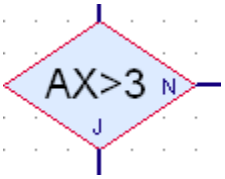
Depending on which interface is selected, the toolbar button changes its appearance. This button and the corresponding menu **Environment** change neither the current ROBOPro program nor which kind of interface is connected to the computer (this is adjusted via the **COM/USB** button).

This button only determines which options are shown in the property windows of program elements. Below you see the property window for a branch for the ROBO Interface and the ROBO TX Controller.





Furthermore, input elements which don't match the selected interface are drawn with a red border.



Most usually you will work in the environment, which matches the interface you own. But there are a few exceptions:

- You want to develop a program, which uses a ROBO TX Controller as well as a ROBO Interface. In online mode this is possible. In this case you develop the program parts intended for the different interfaces in the respective environment. You can change the environment hence and forth any time.
- You own a ROBO TX Controller and got a program, which has been designed for the ROBO Interface, from a friend. If the input configuration is compatible, you can use interface independent programming (see the next section for further information). If you only want to do small changes, it is better to stay in the ROBO Interface environment.
- You own a ROBO TX Controller and want to write a program for a friend, who owns a ROBO Interface. In this case you can use interface independent programming as well and develop the program in the ROBO Interface environment.
- The two above points apply of cause also the other way around.

11.3 Interface independent programming

As long as your program uses inputs only, which are available on the ROBO Interface as well as on the ROBO TX Controller, you can use your program without changes on the ROBO TX Controller as well as on the ROBO Interface. The input mapping is as follows:

ROBO Interface	ROBO TX Controller
D1 (ultrasonic)*	I1 (ultrasonic)*
A1 (analog 10V)	I2 (analog 10V)
AX (analog 5kOhm)	I3 (analog 5kOhm)
AY (analog 5kOhm)	I4 (analog 5kOhm)
I1-I4 (digital)	I5-I8 (digital)
I5-I8 (digital)	C1D-C4D (digital, not for trail sensor)**

**Note:* Only the ultrasonic sensor with 3 connectors and order number 133009 can be attached to the ROBO TX Controller. The ultrasonic sensor matching the ROBO Interface has 2 connectors and order number 128597.

***Note:* The abbreviation C1D means, that the counter input C1 is used as simple digital input. If C1 is used as fast counter input, the window shows C1C.

If your program uses only the inputs listed above, and if the input mode for the universal inputs **I1-I8** matches on the ROBO TX Controller as well, you can load your program on the ROBO Interface as well as on the ROBO TX Controller. The mapping is done automatically, if you start the program

in online or download mode. So you can develop a program in ROBO Interface mode using ROBO Interface inputs, but select via COM/USB a ROBO TX Controller.

11.4 Conversion of programs

If you cannot or do not want to do interface independent programming, you can also make the adaptions to the interface in the program permanently. The menu point **Environment / Map inputs** adjusts all inputs to the selected environment in the way listed in the table above. Inputs, which are not assigned in the table (D2, A2, AV), are not mapped and can be mapped manually later. You can undo this operation by switching the environment and calling the menu function again.

11.5 Universal inputs, sensor type and input mode

With the ROBO Interface, each input has a fixed input type. To the AX input, only resistive sensors can be attached. The ROBO TX Controller, by contrast, has 8 universal inputs I1-I8, which can be controlled by a ROBOPro program such that different sensor types can be attached. The input mode is selected automatically with the sensor type. In older versions of ROBOPro it was possible as well to select a sensor image for each input, but this had only an illustrative purpose and no technical function. With the ROBO TX Controller it is important that you select the right sensor type. Otherwise the input is not configured correctly.

In Level 4 and above, you can also change the input mode independently from the sensor type.

With the ROBO TX Controller, some sensors require different input modes, although they could all be attached to the I1-I8 inputs of the ROBO Interface. This applies mainly to the track sensor, which requires the 10V digital input mode when used with the TX Controller. During conversion of programs and when doing Interface independent programming, ROBOPro uses the previous sensor image as sensor type in order to select the correct input mode.

11.6 Fast counter inputs and extended motor control

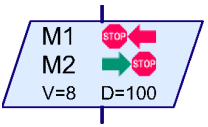
The TX Controller has 4 fast counter inputs C1-C4 and an integrated motor controller, which allow for precise motor control. The extended motor control offers two functions, automatic brake after a specified distance and synchronization of two motors. The motor control system requires that the rotary encoder of motor M1 is connected to fast counter input C1 and so forth.

When using the ***automatic brake*** mode, a number of pulses is specified and the control system breaks the motor automatically when the target is reached. The control system also calculates the braking distance of the motor and starts breaking early enough, so that the chosen distance is reached exactly even with fast motors and high resolution rotary encoders.

When using the ***synchronization***, two motors are controlled such that they make the same number of turns in the same time. This is most useful for track vehicles, which are going exactly straight this way. If one motor becomes slower, the motor control system automatically slows down the other motor.

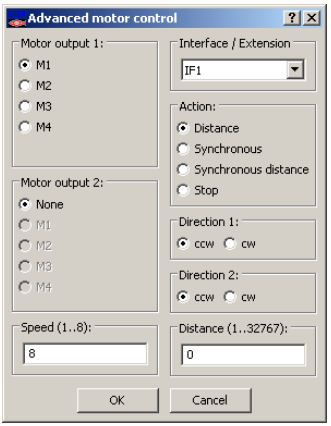
You can also combine these two functions and let the motors go a defined number of pulses with synchronized speed.

11.6.1 Encoder Motor (Level 1)



To comfortably control motors with built-in pulser (encoder), the new programming element Encoder motor is available in level 1 and above.

Using this element, you can either move one motor a defined number or pulses, or two motors synchronized, with or without a predefined number of pulses. The program element offers the following control options:

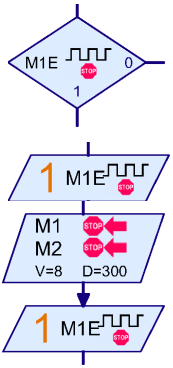


If you want to move only one motor with a defined number of pulses, you choose the **action distance** and enter the desired **speed**, **direction** and **distance**.

Using **action synchronous**, you can move two motors speed synchronized. You can independently select the **direction** for both motors. The **speed** is selected only once for both motors, since both motors shall turn equally fast.

The **action synchronous distance** combines, as explained above, a defined number of pulses with speed synchronization of 2 motors.

With **action stop**, you can stop both motors any time and also end speed synchronization and clear a remaining distance, if applicable. In case you start motors using this element, you also have to stop them again with this element before you can use the usual motor control elements again.



If you defined a distance, the element ***does not wait*** until the chosen distance is reached, but immediately continues with the next program element. This way the program can continue and stop the motors in case of some event. In order to test if the motor has reached its target, there are internal inputs **M1E** to **M4E**, one for each motor. You can query these inputs using a branch or a **wait for input** program element.

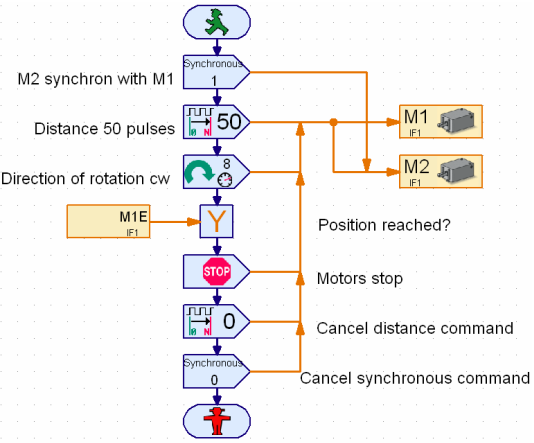
The inputs **M1E** to **M14** become ***1***, if the corresponding motor had reached the given number of pulses (distance). The inputs stay 1 until you send a new distance command for the motor. For the wait element it is therefore best to wait for 1 as in the picture. If you control two motors synchronously, you need a wait element for the first motor only. In the case of synchronously controlled motors the inputs become 1 when both motors have reached their destination.

An example for using this element is shown in the section 4.4 *Tango* on page 3.

11.6.2 Extended Motor Control in Level 3

In level 3, a motor is controlled by sending commands to an orange motor element.

Using the command **synchronous**, a motor can be synchronized with another motor. If, for example, you send the command synchronous with value 1 to motor 2, motor 2 is synchronized with motor 1. In level 3, you can also synchronize more than two motors. The synchronization is cancelled, if you send to a motor the command synchronous with value 0.



With the **distance** command, you can define a number of pulses to go for a motor. As soon as this number of pulses is reached, the motor breaks. The defined distance can be cancelled at any time by sending a distance command with value 0.

If you want to combine synchronization with distance, you must send the distance command to both motors. The Synchronous command is send to one motor only, though, with the number of the other motor as command value.

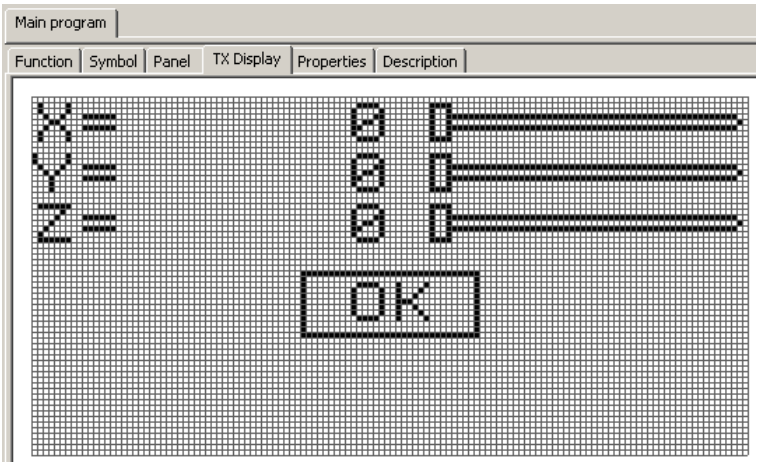
Neither the synchronous nor the distance command actually starts the motor. For this you need a left, right or = command.

You wait for a target reached condition in the same way as in level 1. Of course there are also level 3 elements for the target reached inputs.

After this, to control the motor with normal motor commands again, first you have to cancel the distance and synchronous command again by sending a distance and a synchronous command with the value 0. But ***previously***, you must have sent the motor a stop command. The distance and synchronous command only stop the motor as long as the commands are active. If you cancel the commands without stopping the motor first, the motor carries on running.

11.7 Display

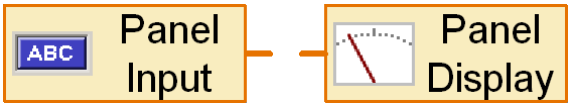
Another new feature of the ROBO TX Controller is the ***display***. Similar to an operation panel, the display can be used to control a program or to output status data. A display is designed in the **TX Display** tab in the same way as on operation panel:



The available control elements are also the same as for an operation panel: a slider and a push button. For displaying status data, a text display is available. For structuring the display area, there is a line element and a rectangle element.

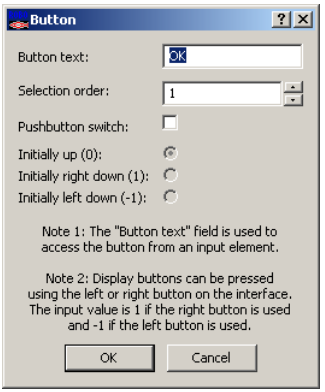
If you want to change the size of a control element, you can use the menu option **Draw / Edit**.

The connection between display elements and the program is done in the same way as for an operation panel using input and output elements.



The display controls are operated via the two buttons on the interface. You can select different control elements by pressing the left or right button shortly. If you press the left or right button longer, the control element is modified. A slider button slides, a push button is pushed.

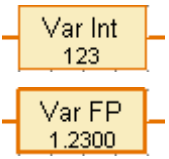
You need to assign in the property window of each control element a selection order number. These numbers define in which order the control elements are selected using the buttons.



Important note: If you want to stop a program in download mode which uses the display functions, you have to press both buttons simultaneously.

In the same way as every subprogram can have its own operation panel, every subprogram can have its own display content. But there is a difference: the display contents changes automatically if a subprogram is entered or left. This way it is possible to develop quite complex menu structures without much effort. It is advisable to run all subprograms with display contents in a single process. Otherwise it might become difficult to predict which display contents is shown in what situation.

12 Working with decimals

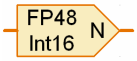


ROBOPro offers in version 2 the option to work with decimals (also called decimal fractions). This means that you can perform arithmetic operations not only with integer numbers like 1 or 2, but also with fractional numbers like 3.99482925 with a precision of 9 digits. The ROBO TX Controller uses so-called floating point arithmetic to implement this feature. It can be used with the ROBO TX Controller in online and download mode. The ROBO Interface supports floating point arithmetic in online mode only at present. The download mode support is planned for the ROBO Interface, though, and will be available in the near future. Also planned is the support for trigonometric and other non rational functions (such as sin, cos, tan, ln, e^x, $\sqrt{}$, x^2).

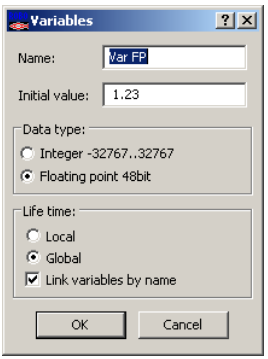
If you are interested in the details: The precision of arithmetic operations is 48 bits with a 32 bit mantissa. This corresponds to a precision of slightly more than 9 decimal digits.

In Version 2.1.1.0, the following functions are supported:

- floating point variable
- floating point list
- operators +, -, *, /
- conversion integer / floating point and vice versa

  This element is located in the element window with the operators.
- branch element which compares a floating point and a constant
- text command with floating point formatting



There are no special floating point elements. Instead you can switch the data type in the property window of the corresponding integer element. Floating point elements are displayed with a thick border.

12.1 Comparing floating point numbers

There is no 3-way comparison element for floating point numbers. The reason is that floating point numbers should not be compared for equality, because the value of a floating point number is usually not exact cause of round of errors. For example with floating point numbers, the result of 10· 0.1 is not equal to 1, because 0.1 cannot be represented exactly with binary floating point numbers.

You can compare a floating point number with a floating point constant using the level 3 comparison element. In ROBOPro version 2.1.2 and above, there will be comparison operators in addition.

12.2 Displaying floating point numbers

Since there is not as much space on the TX Display as on a computer monitor, ROBOPro offers some options to display floating point numbers in a space saving way. The exponent is typically

displayed using the exponent notation common in technology, for example k for thousand as in km. The exponent abbreviations are as follows:

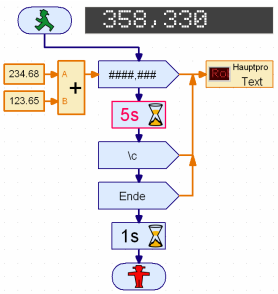
Abbreviation	Name	Exponent
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
u	mikro	10^{-6}
m	milli	10^{-3}
k	kilo	10^{3}
M	Mega	10^{6}
G	Giga	10^{9}
T	Tera	10^{12}
P	Peta	10^{15}
E	Exa	10^{18}

In case the exponent is outside of this range, which mostly happens in case of calculation errors, the error **?FORMAT?** is displayed.

Of cause floating point numbers can be displayed using the notation more common with computers and pocket calculators as well. The text command offers the following options:

Format	Output 1	Output -0.01	Output 1000
####.####	___1.0000	__-0.0100	?FORMAT?
#####	____1	___-0	_1000
##.###^	_1.000	-10.00m	_1.000k
##.###^##	_1.000^00	_1.000v02	_1.000^03
##.#####^^####	_1.0000E+0000	_1.0000E-0002	_1.0000E+0003

Example:



Two constants are added and the result is displayed for 5 seconds. Then the display is erased (entry of \c in the text command) and the word “End“ is displayed.

Please note the following hints regarding formatting:

- The number of valid digits as well as the number of digits in the exponent can be varied in all formats.
- You can use a point or comma as decimal separator.
- In front of the point or comma, at least 2 # characters are required, one for the sign and for the at least 1 digit in front of the decimal separator.

ROBOPro uses the following codes to display special values and to flag error situations:

- **0** is used to represent an exact zero (no error) or numbers that are less than approximately $\pm 10^{-2500}$.
- **?FORMAT?** The number cannot be displayed using the chosen format.
- **?OVERFLOW?** The calculation resulted in an arithmetic overflow. For example, division by zero results in an overflow.
- **?NAN?** Not A Number is the result of invalid calculations like square root of -1
- **?UNDEFINED?** This value is e.g. used for subprogram inputs before they receive a value.
- **?LOST?** is displayed for entries like 0/0 etc.
- **?CORRUPTED?** This should never happen. If you have a program that shows this value, please send it to the fischertechnik service.
- **??.??** See next section.

12.3 Calculation of Precision

In contrast to most other floating point systems, ROBOPro calculates in each operation also the number of valid digits (or bits). Digits that were lost during computation are displayed in the text output as “?“. For example, in ROBOPro the calculation 1.00000001 - 1.00000000 yields the result 9.8??n. The difference would be exactly 0.00000001 or 10n. However, ROBOPro displays one digit more than it can compute exactly. The last digit merely helps determine whether the value should be rounded up or down, in this case from 9.8n to 10n. If you compute 1.0-1.0 in ROBOPro the result is ??.??p. This means 0 with a precision of about 99.99p or 100p, that is 10 to the power of -10. As previously mentioned, an exact 0 (without error) exists, but it is unusual.

13 Connecting more than one ROBO TX Controller to one PC

If you would like to control multiple ROBO TX Controllers from a ROBO Pro program, you no longer need to switch every device to a “unique serial number” and connect it separately to the PC with a USB cord as you did with the earlier ROBO Interface. Instead, only one ROBO TX Controller is connected to the PC via USB. This device is called “master”. Further devices are connected to the ports EXT1 or EXT2 of the master as so-called “extensions”. The procedure is explained in detail in the chapter on “Extensions“ in the operating manual of the ROBO TX Controller. Up to 8 devices can be connected to a master in this manner. The data of all devices is transmitted bundled and very efficiently to the PC via one single USB port.

The connection of multiple ROBO TX Controllers to multiple USB ports of one PC is not supported.