

# TABLE OF CONTENTS

<b>1 Introduction – controlling fischertechnik models with ROBO Pro</b> .....	<b>91</b>
1.1 Installation of ROBO Pro.....	91
1.2 Installing the USB driver for the ROBO Interface.....	92
1.3 First Steps.....	95
<b>2 A quick hardware test before programming</b> .....	<b>98</b>
2.1 Connecting the Interface to the PC.....	98
2.2 Getting the right connection – Interface settings.....	98
2.3 Wrong connection: no connection to the Interface!?	100
2.4 Is everything working – the Interface test.....	100
<b>3 Level 1: Your first control program</b>	<b>102</b>
3.1 Creating a new program.....	102
3.2 The elements of a control program ...	103
3.3 Inserting, moving and modifying program elements.....	103
3.4 Linking program elements.....	106
3.5 Testing your first control program .....	107
3.6 Other program elements .....	108
3.6.1 Time delay .....	108
3.6.2 Wait for input.....	109
3.6.3 Pulse counter .....	109
3.6.4 Counter loop .....	110
3.7 Online and download operation—what's the difference?.....	110
3.8 Tips and Tricks.....	112
<b>4 Level 2: Working with subprograms</b> .....	<b>114</b>
4.1 Your first subprogram.....	115
4.2 The subprogram library.....	119
4.2.1 Using the Library.....	119
4.2.2 Using your own library .....	119
4.3 Editing subprogram symbols.....	120
<b>5 Level 3: Variables, panels &amp; Co</b> .....	<b>122</b>
5.1 Variables and commands.....	122
5.2 Variables and multiple processes .....	123
5.3 Panels .....	124
5.4 Timers .....	127
5.5 Command inputs for subprograms....	128
5.6 Lists (Arrays).....	131
5.7 Operators .....	132
<b>6 Extension modules and controlling several interfaces</b> .....	<b>136</b>
6.1 Extension Modules.....	136
6.2 Multiple Interfaces.....	136
6.3 Interface assignments in subprograms .....	138
6.4 Tips and Tricks.....	138
6.5 Changing the Interface serial number or firmware version.....	139
<b>7 Program element overview</b> .....	<b>141</b>
7.1 Basic elements (Level 1).....	141
7.1.1 Start .....	141
7.1.2 End.....	141
7.1.3 Digital Branch.....	141
7.1.4 Analog Branch .....	142
7.1.5 Time delay .....	142
7.1.6 Motor output.....	143
7.1.7 Lamp output (Level 2).....	144
7.1.8 Wait for input.....	145
7.1.9 Pulse counter .....	146
7.1.10 Counter loop .....	146
7.2 Subprogram I/O (Level 2-3).....	147
7.2.1 Subprogram entry (Level 2).....	147
7.2.2 Subprogram exit (Level 2) .....	147
7.2.3 Subprogram command input (Level 3).....	147
7.2.4 Subprogram command output (Level 3) .....	148
7.3 Variable, List, ... (Level 3) .....	148
7.3.1 Variable (global).....	148
7.3.2 Local variables .....	149
7.3.3 Constant.....	150
7.3.4 Timer variable .....	150
7.3.5 List .....	151
7.4 Commands (Level 3).....	152
7.4.1 = (Assignment).....	153
7.4.2 + (Plus) .....	153
7.4.3 – (Minus).....	154
7.4.4 Right.....	154
7.4.5 Left .....	154
7.4.6 Stop.....	154
7.4.7 On .....	154
7.4.8 Off .....	154
7.4.9 Text.....	155

7.4.10	Append value .....	155
7.4.11	Delete value(s) .....	155
7.4.12	Exchange values .....	155
7.5	<i>Compare, wait for, ... (Level 3)</i> .....	155
7.5.1	Branch (with data input) .....	155
7.5.2	Comparison with fixed value .....	156
7.5.3	Compare .....	156
7.5.4	Time delay .....	156
7.5.5	Wait for .....	157
7.5.6	Pulse counter .....	157
7.6	<i>Interface inputs/outputs</i> .....	158
7.6.1	Digital input .....	158
7.6.2	Analog input .....	159
7.6.3	IR Input .....	160
7.6.4	Motor output .....	161
7.6.5	Lamp output .....	161
7.6.6	Panel input .....	162
7.6.7	Panel Output .....	163
7.7	<i>Operators</i> .....	163
7.7.1	Arithmetic operators .....	164
7.7.2	Logical operators .....	164
<b>8</b>	<b>Panel elements and panels: overview</b> .....	<b>166</b>
8.1	<i>Displays</i> .....	166
8.1.1	Meter .....	166
8.1.2	Text display .....	167
8.1.3	Display lamp .....	168
8.2	<i>Control elements</i> .....	169
8.2.1	Button .....	169
8.2.2	Slider .....	169
<b>9</b>	<b>Drawing functions</b> .....	<b>171</b>

# 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 Interface (item number 93293) and the earlier Intelligent Interface (item number 30402) are available from fischertechnik. You can use either Interface with ROBO Pro. But ROBO Pro only supports the online mode of the Intelligent Interface. ROBO Pro no longer supports the old 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.

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 processor with a clock speed of at least 600 MHz, 32 MB RAM and ca. 20 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 95, 98, ME, NT4.0, 2000 or XP
- A free USB interface or a free RS232 interface (COM1 to COM4) to connect the ROBO Interface (item number 93293); or a free RS232 interface (COM1 to COM4) to connect the earlier Intelligent Interface (item number 30402).

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 suc-

cessful 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:\Programs\ROBO Pro. 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 for the ROBO Interface

This step is only required if the new 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. This step is not required for the earlier Intelligent Interface, since the Intelligent Interface can only be connected to a serial port. 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.

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 this case, however, you will only be able to connect your Interface via the somewhat slower serial port. Important note for installation under Windows 2000 and Windows XP:

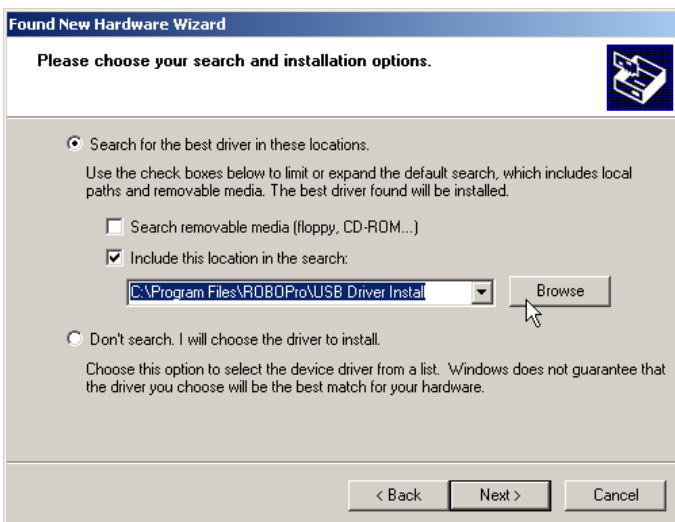
In order to install the USB driver, you must first connect the ROBO Interface with a USB cable to your computer and supply it with power. Windows recognizes automatically that the Interface is connected and displays the following window:



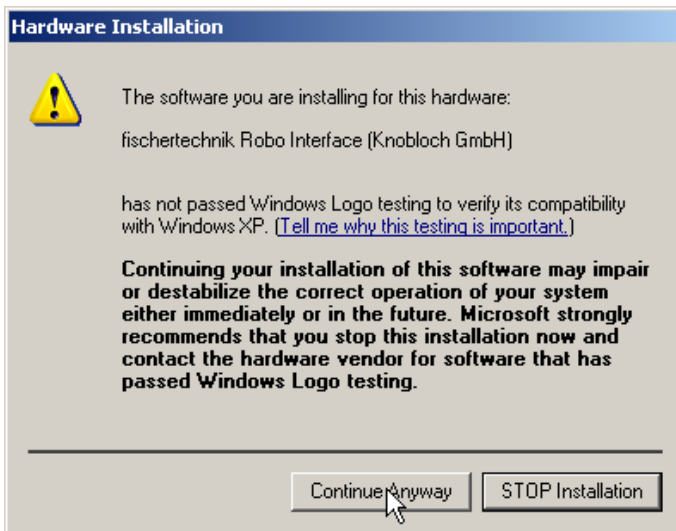
Depending on the operating system, the appearance of the window can be somewhat different from the illustration above!

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:\ROBOPro):

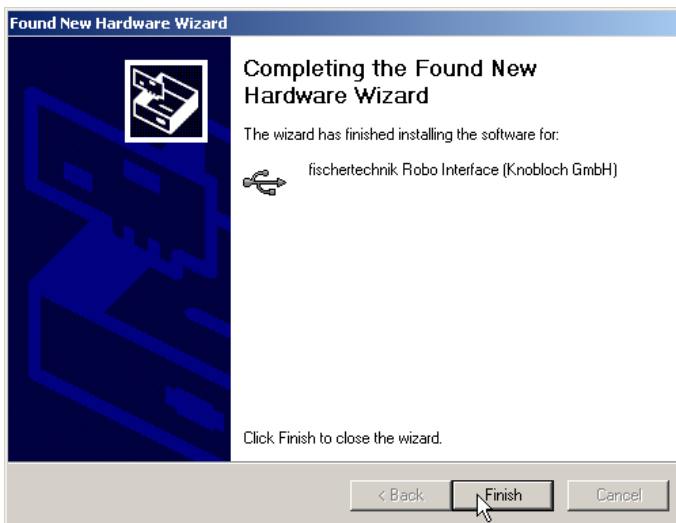


Under Windows XP, you may see the following message after pressing **Next**:



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**.

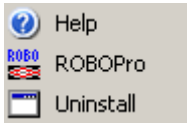
Finally, the following message will appear:



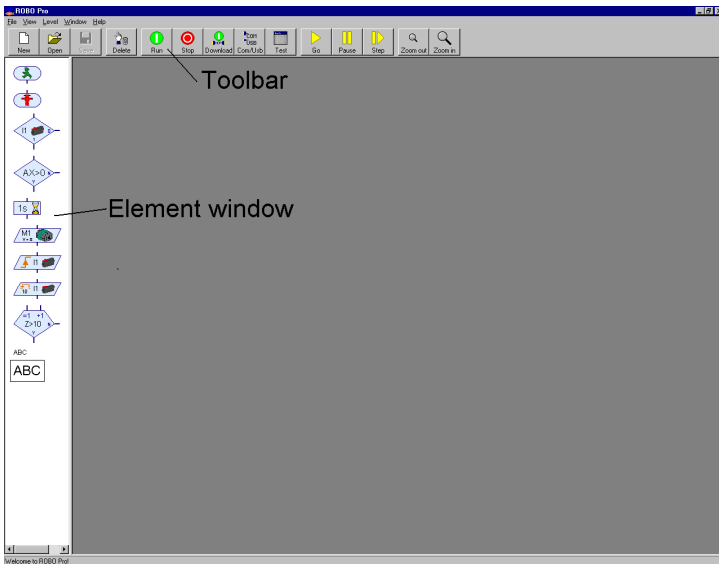
Press **Finish** to complete USB driver installation.

## 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.

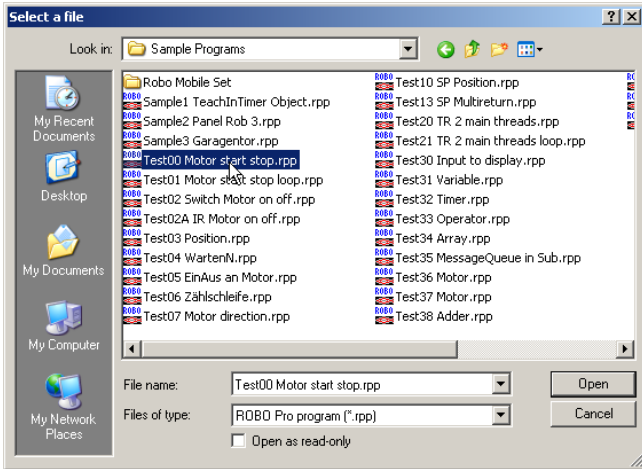


The window has a menu bar and toolbar with various operating buttons above as well as a window on the left-hand side with program elements. If you see two stacked windows in the left margin, ROBO Pro is not set on **Level 1**. To allow the functionality of ROBO Pro to match your growing knowledge, you 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.

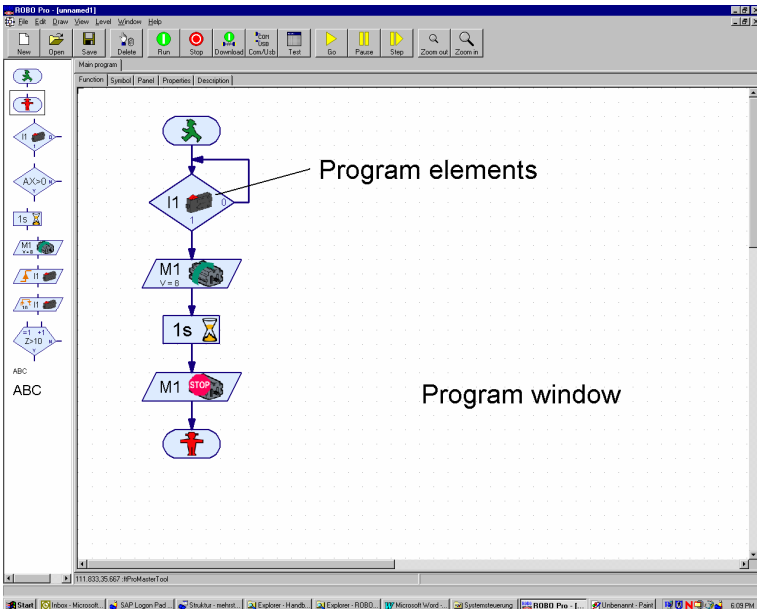


Open

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:\Programs\ROBO Pro\Sample programs**.



Open the file **Test00 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 Interface item number 93293 or the earlier Intelligent Interface item number 30402), appropriate software settings must also be made and the connection tested. We will 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:

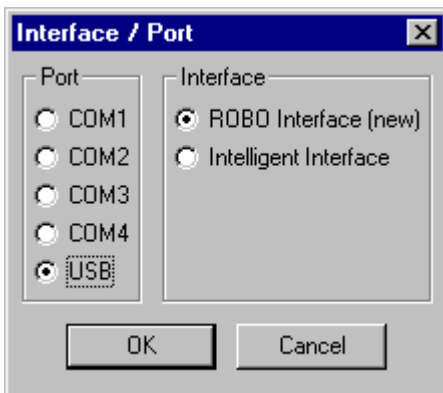
- With the ROBO Interface (item number 93293) either a USB port or one of the serial ports COM1 to COM4 can be used.
- With the ROBO Interface (item number 30402) one of the serial ports COM1 to COM4 must be used.

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 Interface.

### 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:

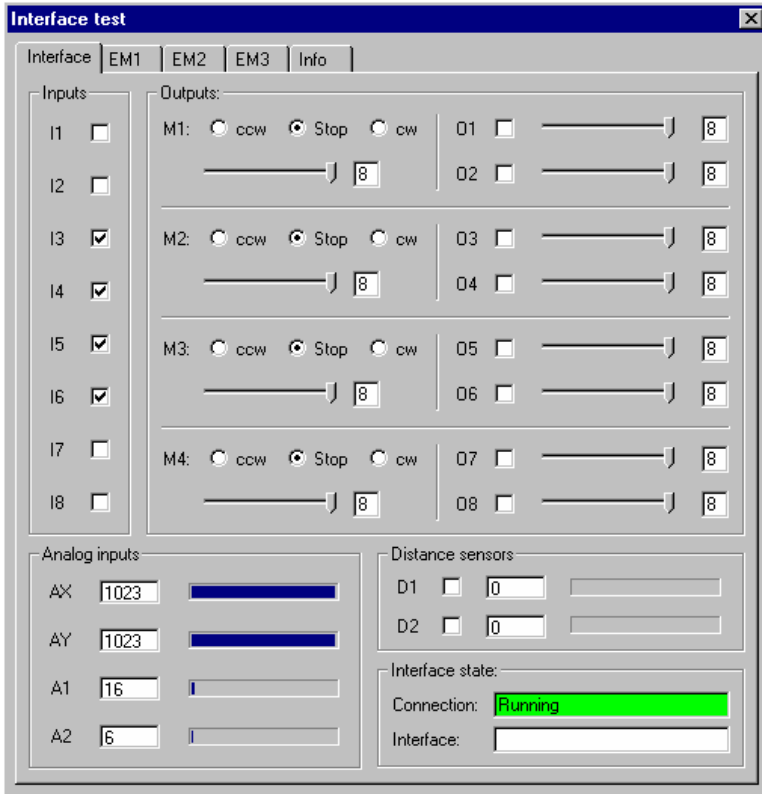


Here you can select the port as well as the Interface type. As already mentioned, the Intelligent Interface supports only the serial ports COM1—COM4.

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.



Test



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:

- **Interface connection OK** confirms correct connection to the Interface
- **No connection to Interface** 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!?

With either ROBO or Intelligent Interface, if you get the message **No connection to Interface** despite having correctly set the serial COM 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 Interface's processor stops working. In this case either the red LED lights up continuously or no LED lights up. If the voltage is correct, some of the green LEDs will flash on and off. With the earlier Intelligent Interface it is not easy to determine from the LEDs whether there is sufficient voltage for the processor. 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.
- **Is the port actually working?**  
You can find this out by testing another serial device, such as an external modem, on the port.
- 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).
- Only for Windows NT/2000/XP and the earlier Intelligent Interface: If an earlier Intelligent Interface is already connected with the computer and power supply at PC boot-up time, Windows NT will unfortunately switch it into download mode. To re-establish the connection with the PC, you merely have to interrupt the power supply to the Interface briefly. This problem cannot arise with the new ROBO Interface.
- 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, we can use the Interface test to test the Interface itself and the models connected to it. As already mentioned, the test window displays the various inputs and outputs of the Interface:

- **Digital inputs I1—I8**  
I1—I8 are the digital inputs to the Interface. This is where sensors are connected. Digital inputs can only assume the states 0 and 1, or Yes and No. Switches (mini pushbutton-switches), as well as phototransistors (light sensors) or reed-switches (magnetic sensors), can be connected as 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.
- **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 also be controlled in 8 steps as to speed, as well as in direction. Speed is controlled using the slide control. The speed is also 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 the ground socket of the Interface (⊥).
- **Analog inputs AX–AY**  
The analog inputs AX and AY measure the resistance of the connected sensor. Here NTC resistors to measure temperature, potentiometers, photoresistors or phototransistors can be connected.
- **Analog inputs A1–A2**  
These two inputs measure a voltage of 0–10V.
- **Distance sensors D1–D2**  
Only special distance sensors may be connected to the distance sensor inputs D1 and D2. The distance sensors D1 and D2 are available both as digital and as analog inputs.
- **Extension modules EM1–EM3**  
Where extension modules are connected to the Interface (up to three I/O Extensions item number 93294 for the ROBO Interface, but a maximum of one Extension Module item number 16554 for the Intelligent Interface), these can be controlled by switching to the extension modules by means of the register on the upper edge.

### 3 Level 1: Your first control program

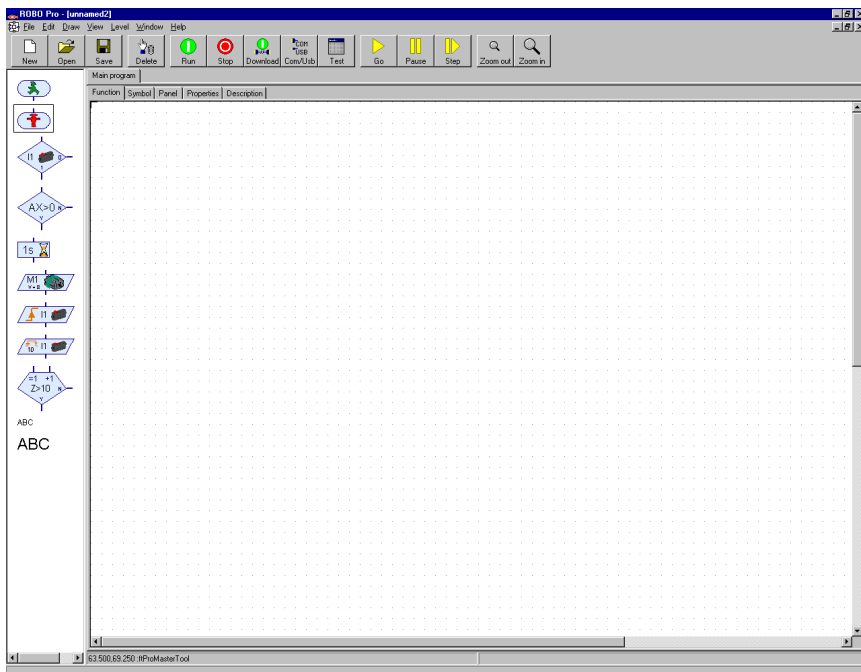
After testing the hardware, that is the Interface and the switches and motors connected to it, in Chapter 1, we'll now get down to programming. But what does "programming" actually mean? Well, just imagine that, for example, a robot is connected to our Interface. But this robot is so stupid that it can't 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



New

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.

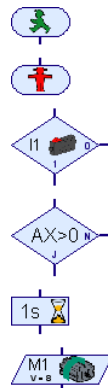
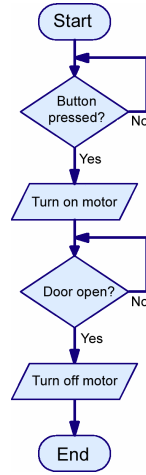
You put the flow chart together from program elements. Another new concept? Don't worry! In ROBO Pro the individual elements that are put together to form a flow chart are called program elements. The action "switch on motor" means just that: the Interface should actually switch on the motor that is 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

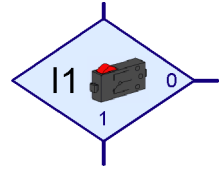
Now it's a matter of creating a flow chart for our garage door control system from the program elements contained in the element window. All available program elements can be fetched from the element window and inserted in the program window.

#### 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

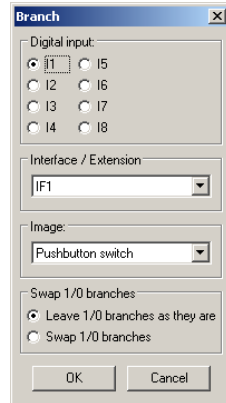
It is quite simple to delete elements. You can delete all the elements marked in red (see previous section) by pressing the “delete” key (**Del**) on the keyboard. You can also delete individual elements with the Delete function. To do this, first click on the button in the toolbar like the one illustrated 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.

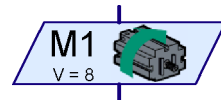
- Buttons **I1** to **I8** allow you to enter which of the Interface's inputs is to be queried.
- The selection **Interface / Extension** is not explained until Chapter 6 *Extension modules and controlling several Interfaces* on page 136
- Under **Image** you can select an image for the sensor connected to the input. Digital inputs are mostly used with push-button sensors, but often also with phototransistors or reed-contact switches.
- 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 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.

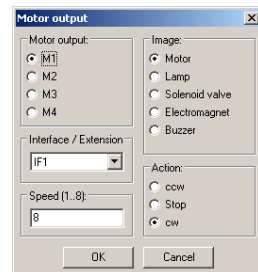
If you connect a mini-sensor as an "opener", using connections 1 and 2 of the switch, the program branches to the "1" branch if the switch is depressed, and otherwise to the "0" branch.

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 fischartechnik component connected to the output.
- We will deal with the selection **Interface / Extension** when we get to Chapter 6, *Extension modules and controlling several Interfaces* on page 136
- 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.



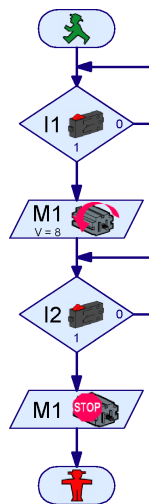
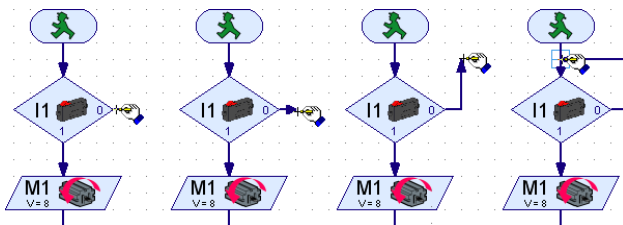
- Under **Speed/Intensity** you can set the speed at which a motor is to operate, or how brightly a lamp should glow. Possible values are 1 to 8.

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: If 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.

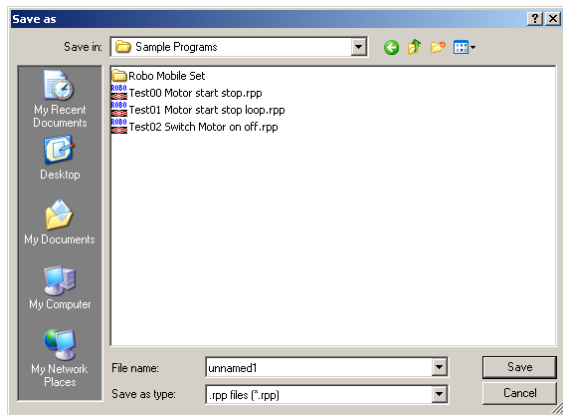
### 3.5 Testing your first control program

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.

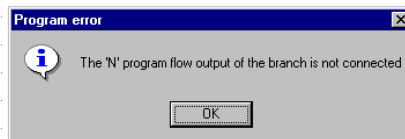
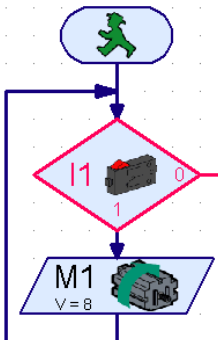
Under “Save in”, choose the directory in which you want to save the file. Under “Filename”, enter a name not yet in use, e.g. GARAGE DOOR and confirm by left-clicking on “Save”.



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:

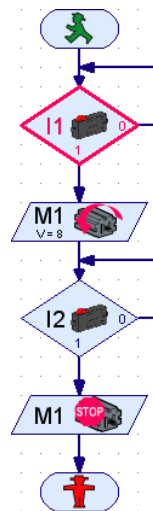


If you have received an error message, you must first of all correct the reported error. If you do not, the program will not be started.

Note: You will find a full explanation of this mode of operation and of “Download Operation” in Section 3.7, on page 110.

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 fulfills 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.

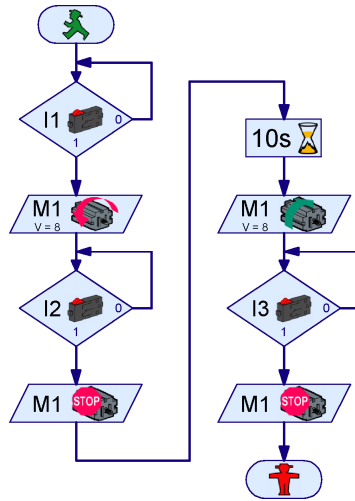
### 3.6.1 Time delay

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.

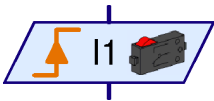




The finished flow chart should look roughly as presented on the right. For the sake of presentation, the new program elements have been moved to the right. Once there are no more mistakes in the flow chart, you can test the extended garage door control system as usual with the **Start** button. The motor 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,

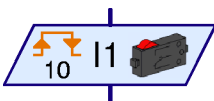
waits until one of the Interface's inputs is in a particular state of has changed in a particular way. There are 5 variants of this element.

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.

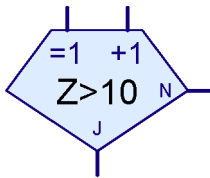
### 3.6.3 Pulse counter



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

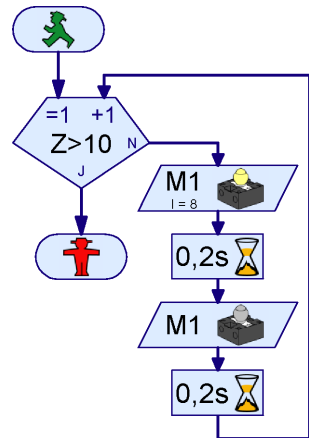
changes are regarded as pulses. With pulse wheels, one normally waits for changes in either direction, so that a resolution of 8 steps per revolution is obtained with 4 gear teeth.

### 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

If you have a ROBO Interface (not an Intelligent Interface), you can also use **download operation** instead of online operation. In online operation programs are executed by your computer. In this mode, your computer 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 Interface. 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 Interface.

With the ROBO Interface the problematic cable can be replaced by the **ROBO RF Data Link**, item number 93295. In that way the model has unrestricted mobility even in online operation.

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. As well as that, several Interfaces can be controlled in parallel in online operation, even a combination of ROBO Interfaces and Intelligent Interfaces.

An overview of the two modes of operation

Mode	Advantage	Disadvantage
Online	<ul style="list-style-type: none"> <li>• Program execution can be followed on screen</li> <li>• Execution, even of large programs, is very fast</li> <li>• Several Interfaces can be controlled in parallel</li> <li>• The earlier Intelligent Interface is supported</li> <li>• Panels can be used</li> <li>• The program can be stopped and continued</li> </ul>	<ul style="list-style-type: none"> <li>• Computer and Interface must remain connected</li> </ul>
Download	<ul style="list-style-type: none"> <li>• Computer and Interface can be separated after download</li> </ul>	<ul style="list-style-type: none"> <li>• The earlier Intelligent Interface is not supported</li> <li>• Program execution cannot be followed on screen</li> <li>• The program can only control up to 3 Extension Modules</li> </ul>

## Using download mode

If you have the new 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.



Download

You can store two separate programs, for example two different behavior modes for a mobile robot, in the two Flash memory zones. You can select, start and stop the two programs using the **Prog** key on the Interface. If the **Start program after download** option is active, the program is started

immediately after download. While the program is running, the green **Prog 1** (Program loaded in Flash 1) or **Prog 2** (Program loaded in Flash 2) LED will blink next to the **Prog** key. If the program has been loaded into RAM, both LEDs will blink. To stop the program, you press the **Prog** key. Then the LED stays alight. To change between Program 1 and Program 2, you hold the Prog key down until the LED for the desired program, 1 or 2, lights up. To start the program, you press the Prog key again.

For mobile robots, the option **Start program with key on Interface** makes more sense. This is because, if you don't have an RF Data Link, you still have to unplug the cable before your program sets the robot in motion. In this case you have to first select the desired program using the Prog key on the Interface, and start the program by pressing it again.

If the last option, **Start automatically on power-up**, is active, the program in Flash 1 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 Prog key every time you switch it on.

#### **Notes:**

When a program is loaded into the Flash memory or executed from Flash memory, programs loaded in RAM are lost, because Flash programs also use RAM memory.

You can also find a comprehensive description of the functions of the ROBO Interface in the operating manual to the Interface.

## **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.

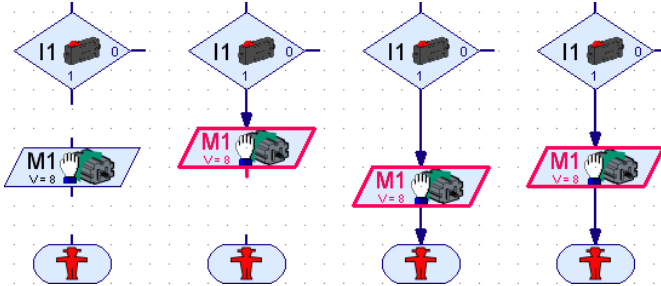


If the mouse is positioned near an end point or a corner of a connecting line, you can move this point while holding down the left mouse key. You can only move a connected line endpoint to another suitable program element connection. In this case the endpoint of the connecting line will be linked to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



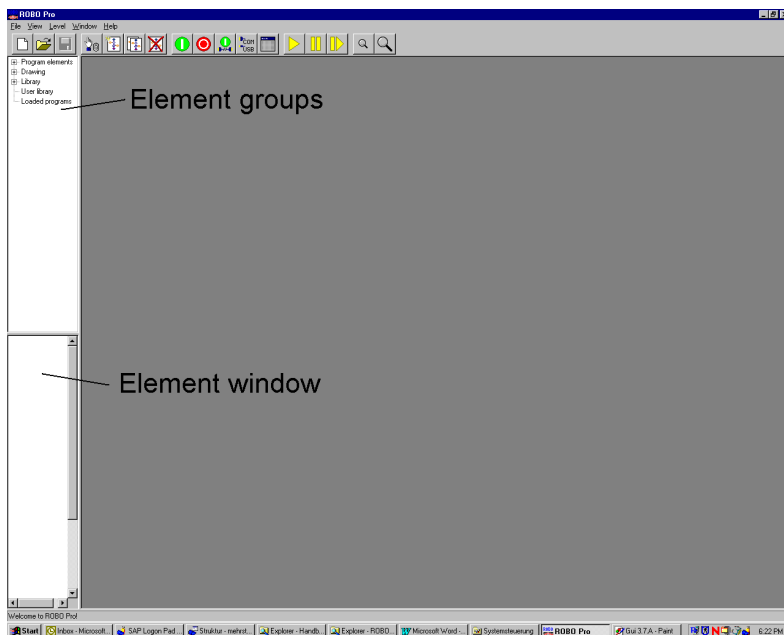
## A different approach to connecting lines

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.

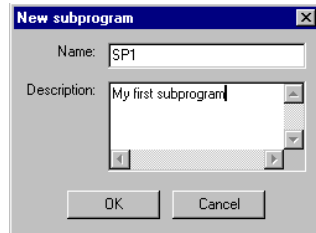
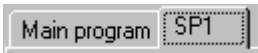
So, now let's get down to the real topic: subprograms! Of course the flow charts we have designed so far have not been on such a large scale that we could not take them all in at once, but surely you can imagine that this could very easily happen in the case of larger projects with more complex flow 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

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 108 ) into subprograms. The program consists of four functional units:

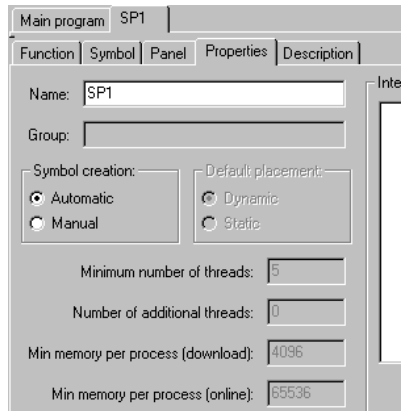
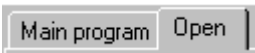
- Wait until button I1 is pressed
- Open door
- Wait ten seconds
- Close door

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.

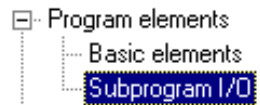



Are you ready to write your first subprogram? Well let's go! But just how does a subprogram start? Good question! You have always begun a main program with the start element. A subprogram begins with a similar element the Subprogram Entry. The element has this name because it is through this element 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.



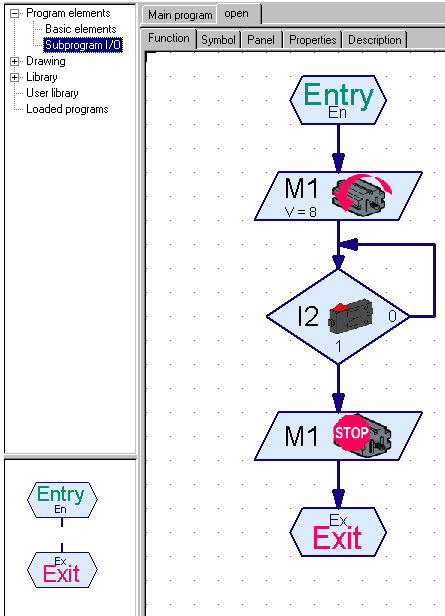
The subprogram now runs identically to the part of the main program which does the opening. You switch on the motor M1 in the left-rotating (anticlockwise) direction, wait until the sensor on input I2 is closed and then switch the motor off again.

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.



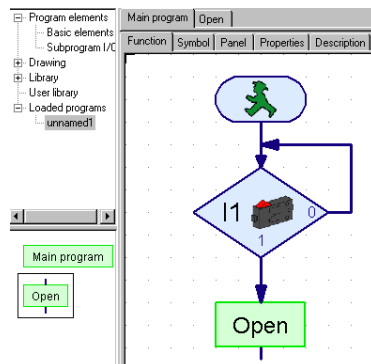
	Stop element	Stops program execution of an independent process
	Subprogram Exit	Here program control is handed back from the subprogram to the main program

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!

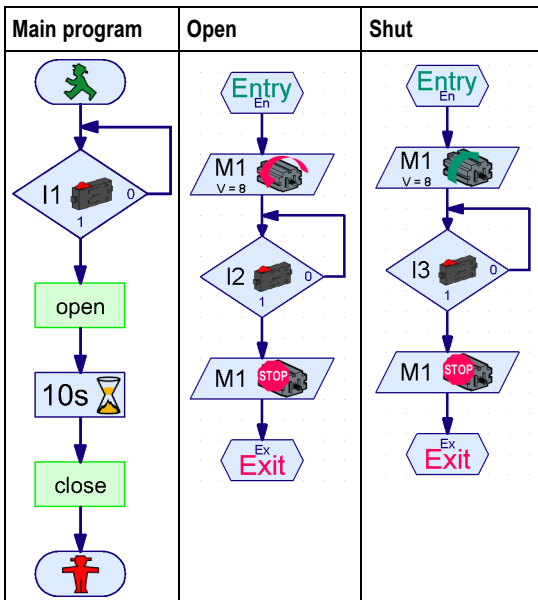


SP New

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:

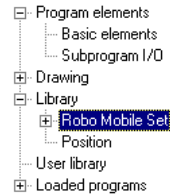




The program starts at the Start element in the **main program**. Then it waits until sensor I1 is depressed. Incidentally, you could also do this using the **Wait for Input** element (see Section 7.1.8 *Wait for input* on page 145 ). After the switch I1 has been pressed, the main program calls the subprogram **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.

Main program		cw 90	
Function	Symbol	Panel	Description
			Turns the robot 90° clockwise.
			The robot turns around on the spot.
			This subprogram requires pulse switches.

**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.

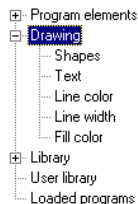
## Organizing your own library

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.

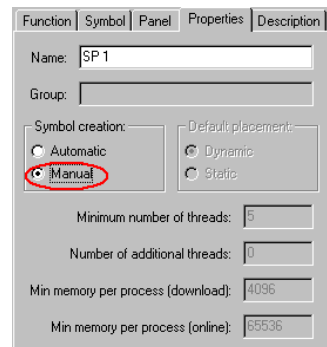
If you have modified a library file and saved it, then you must select the menu item **Update own library** in the **File** menu. This will update the file list in the group window.

## 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 9 *Drawing functions* on page 171. Also observe the functions in the main menu under **Draw**.

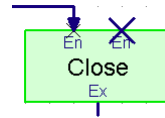


You can also move the connections of the subprogram, but you can't delete the connections or add new ones. In the subprogram symbol there is always one connection for each subprogram



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.

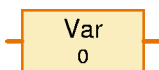


## 5 Level 3: Variables, panels & Co

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

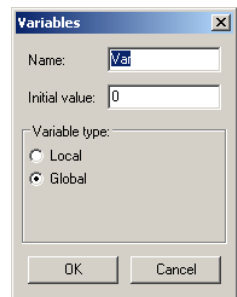
Just imagine that you discover a fascinating machine in a preciously unexplored side corridor of a museum, and you just have to emulate it in fischertechnik. But while investigating the machine you lose track of time, and don't notice that all the other visitors are leaving the museum. Only when the 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

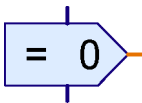


A variable is an element that can hold a number. In the variable Properties window you enter a **Name**, which should give some hint as to what sort of number is stored in the variable. Under

**Initial value** you can specify what number should be stored in the variable at the beginning of the program. The **Variable type** setting will be explained in Section 7.3.2, *Local variables* on page 149.

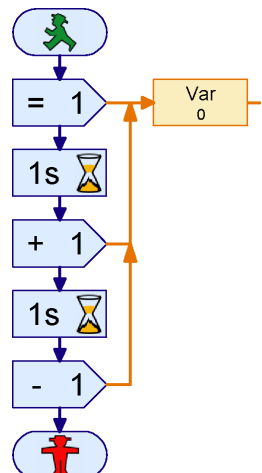


You can alter the stored value by sending commands to the variable. A variable understands three different commands: =, + and -. The = command replaces the stored value with a new value. The + and - commands add something to or subtract something from the stored number.



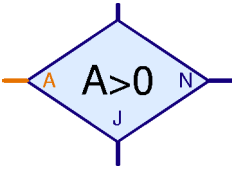
You send the commands to the variable with a **Command element**. Like most other program elements, the Command element has a blue program entry above and a blue program exit below. But to the right it has something quite new, an **orange** connection. That is a command output. Whenever the Command element 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.

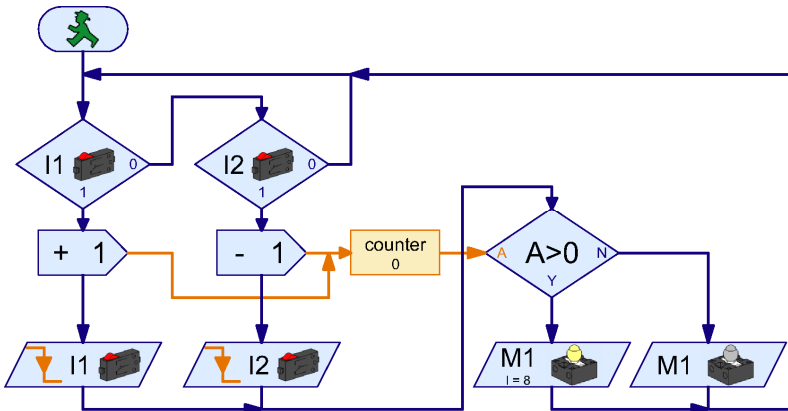
Now try to draw this simple program in ROBO Pro. You will find the command elements in the **Commands** group, the variables in the group **Variable, Timer, ...** If you execute the program in online mode, you will see how the value of the variable changes.



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.

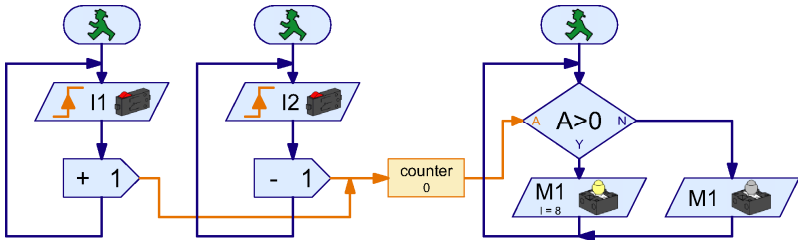


Copy the above program and try it out. As soon as you press the sensor on I1 and release it again, the warning light on M1 lights up. If you operate the sensor on I2, it goes out again. If you operate I1 several times, you must operate I2 the same number of times to make the warning lamp go off again. 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

spective 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.

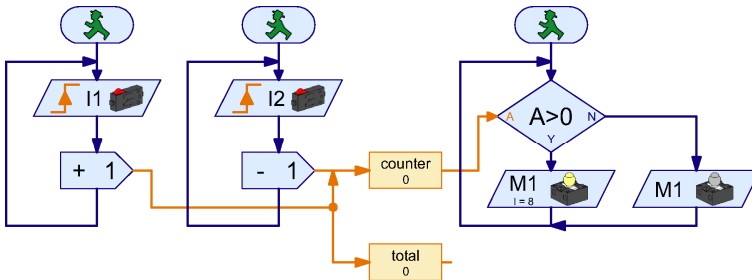
If you have drawn a panel and it has thereafter suddenly disappeared, then presumably you selected a subprogram in the subprogram bar. Switch back to Main program and your panel is sure to be there again.

Var: 0

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.

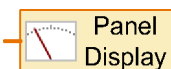
But first you must add to your program a second variable, which will count the number of visitors at the entrance without deducting them from the total again at the exit. In the function bar, you switch back to **Functions**, and insert the variable **Total** as follows:



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.

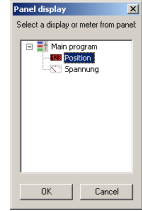


Tip: If orange lines branch, it is often more practical to draw the lines from their target to their origin. If you would like for example to draw the line to the variable **Total**, click first on the input to the variable **Total** and then move the line backwards to the branch point. If, on the other 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.

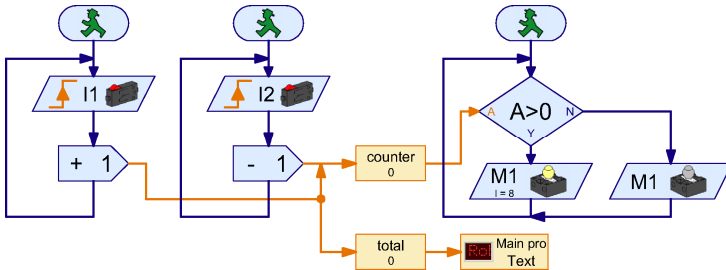
As you will normally have more than one display in a panel, you still need to let the panel output know which display to send the variable values to. This is done quite simply by means of the Properties window of the element. If you right-click on the Panel output element, you will see a list of all 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:



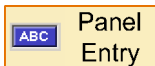
Try it out straight away. As soon as you have started the program in online mode, the display in the panel shows the number of visitors that have passed through the turnstile at the entry.

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.

### Button

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**.

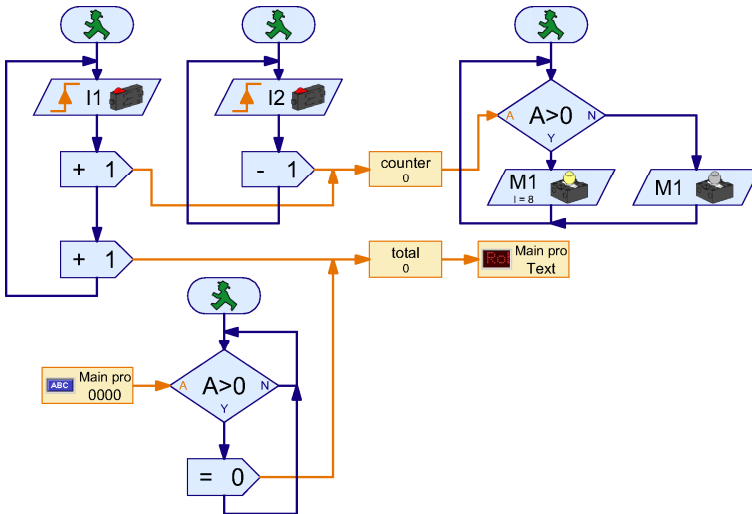
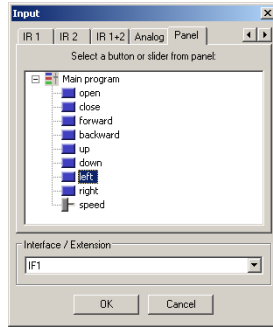


Exactly as in the case of the text display, we also need a program element that will link the button to the program. So start by switching back to **Function** in the function bar. You will find the illustrated **Panel input** element in the **Inputs, outputs** group in the element window. Position it 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 like 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 123.
- 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

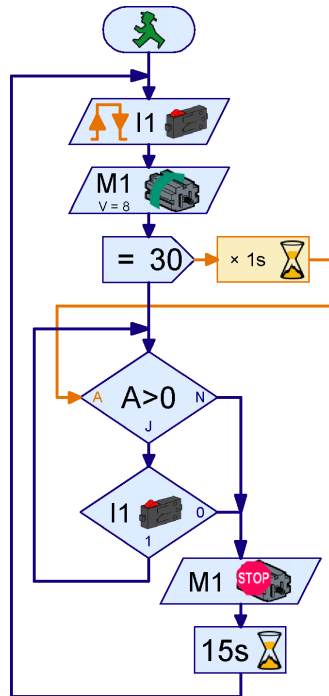
timer keeps track of a number and you can alter the number with =, + and – commands. What distinguishes a timer, however, is that it automatically counts down the number at regular intervals until it reaches 0. The time interval between decrements can be set in steps between one thousandth of a second 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

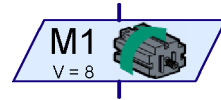
As always, your program works brilliantly, and fischertechnik is pleased, because all the models in the museum are being equipped with the ROBO Interface. Only, like public institutions everywhere, the museum is strapped for cash. So the curator would like to make do with as few Interfaces as possible. But then a ROBO Interface has four motor outputs and also enough inputs to control four



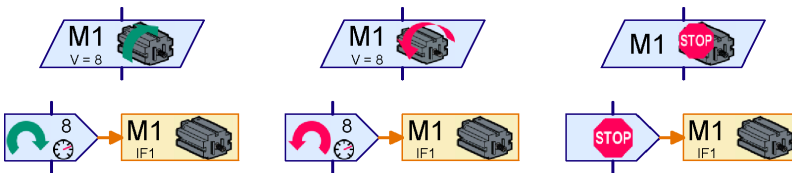
models. As most models can only turn in one direction, you can control as many as 8 models via the single-pole outputs O1 to O8.

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.

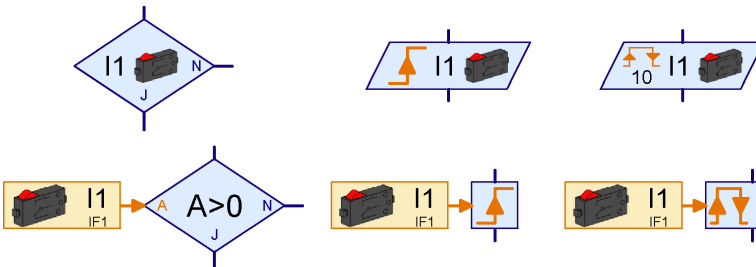


But there is a much more elegant solution to this problem. The trick is to separate the control commands from the motor symbols. Then you can put the control commands (left, right, stop) in the subprogram and the Motor elements in the main program. In the subprogram, using a Command element, which you 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.

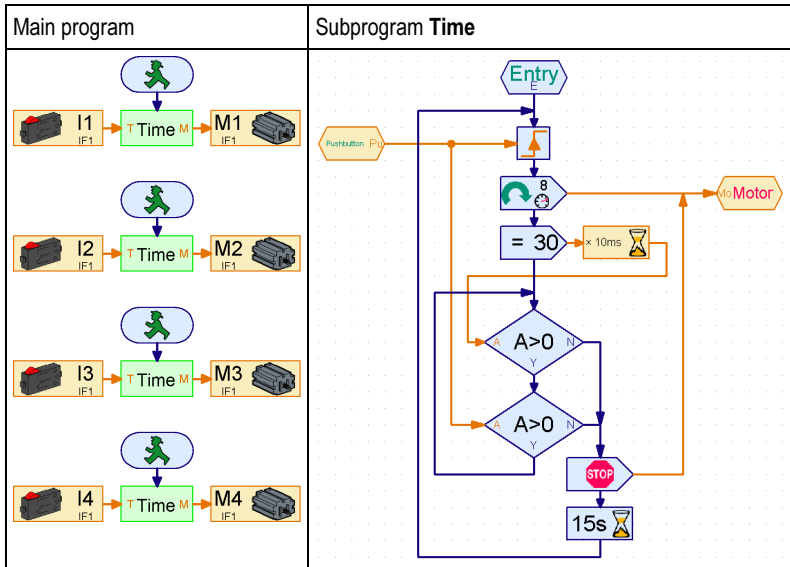


Via a subprogram command input, you can send commands to a subprogram from outside. The Digital input element (sensor) sends its new value over the orange line if the state of the input changes (with what is known as an “= command”). In the element’s dialog field you can give the input a name.



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**.



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 subpro-

gram 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 114.

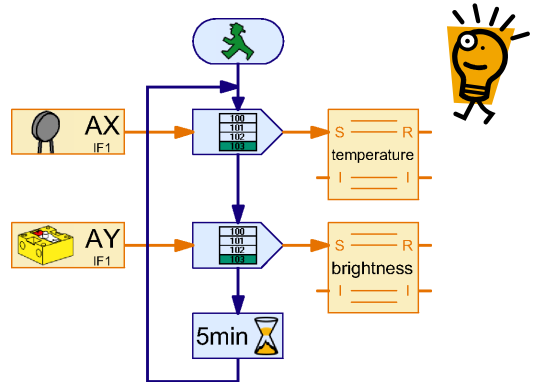
## 5.6 Lists (Arrays)

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.

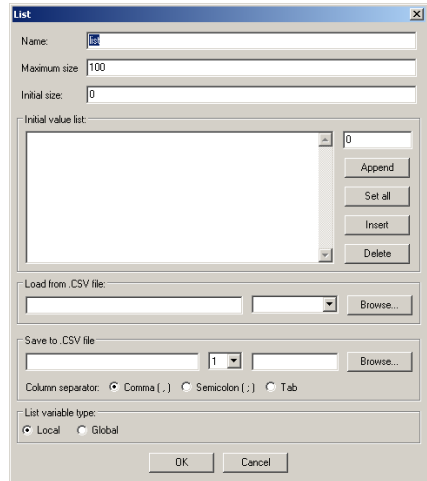
Hint: When inserting the command element you must activate the option Data input for command value in the Properties window. Then a data input will appear on the left of the command element, and you can connect the analog input to it.



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.

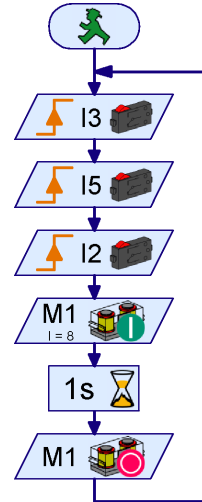
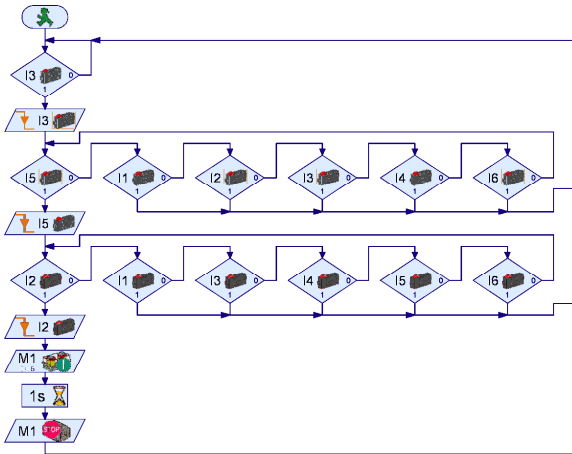
As long as the program is still running in online mode, you can also look at the data in a list by right-clicking on the List element.

## 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!

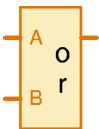
But this problem can be easily avoided with an electronic combination lock. The combination lock is to have a keypad with keys 1 to 6. If three figures are entered correctly one after another, the combination lock should release the climate-control cover by means of a magnet.

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:

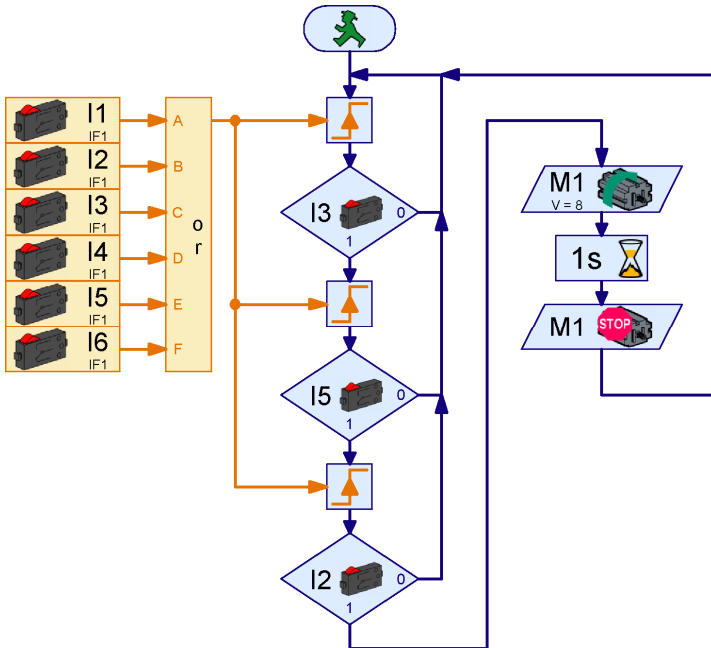


This program opens the lock only when the keys 3-5-2 are pressed without any other key being pressed in between. If for example the key 3 is pressed, the program first waits until the key is released again. If any key other than 5 is pressed next,

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.



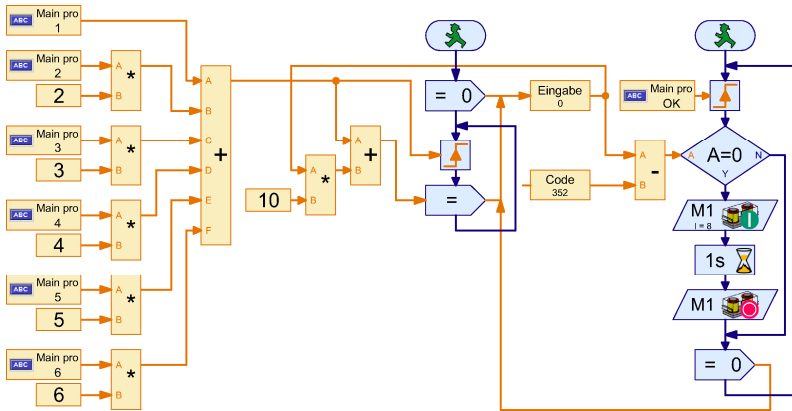
The buttons on inputs I1 to I6 are bundled together via an OR operator with 6 inputs. If at least one of the buttons is pressed, the OR operator yields an output value of 1; otherwise 0. With a **Wait for** element, the program waits until one of the buttons is pressed. Following this, we test immediately 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?



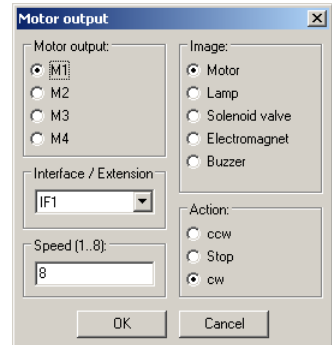
## 6 Extension modules and controlling several Interfaces

One ROBO Interface or Intelligent Interface is enough to control quite resource-intensive models. However, maybe one or another of you likes things a bit more complicated. If you can't get by with 8 sensors, 4 motors and 4 analog inputs, you can enhance your ROBO Interface with up to 3 **ROBO I/O Extensions** or your Intelligent Interface with an **Extension Module**. And for those for whom even that is not enough, there is also the possibility of controlling several Interfaces (each possibly with extension modules) in online mode from your program. So, at least in online mode, the number of inputs and outputs is limited only by the performance of your PC.

### 6.1 Extension Modules

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 actual Interface (i.e. not an Extension Module)
- **EM1..EM3:** These are Extension Modules 1 to 3. If you are using an Intelligent Interface, you can use only one Extension Module (**EM1**). But the Extension Modules **EM2** and **EM3** are also displayed if you are working with an Intelligent Interface, because the programs should be independent of whether you are using a ROBO Interface or an Intelligent Interface.



So it is quite easy to use extension modules like **ROBO I/O Extensions**. You only need to choose the desired module (Interface or extension) for inputs and outputs.

### 6.2 Multiple Interfaces

If you want to control several Interfaces from one program, this only works in online mode. You can, for instance, connect one ROBO Interface to COM1, a second to USB. Each can be fitted with up to 3 ROBO I/O-Extensions. Another possible combination would be for example 2 ROBO Interfaces on USB (both with I/O-Extensions) and an Intelligent Interface on COM1 (maybe with an extension module). So that you can define the intended Interface in the Properties window of an input or output, you have to configure the Interface assignment.

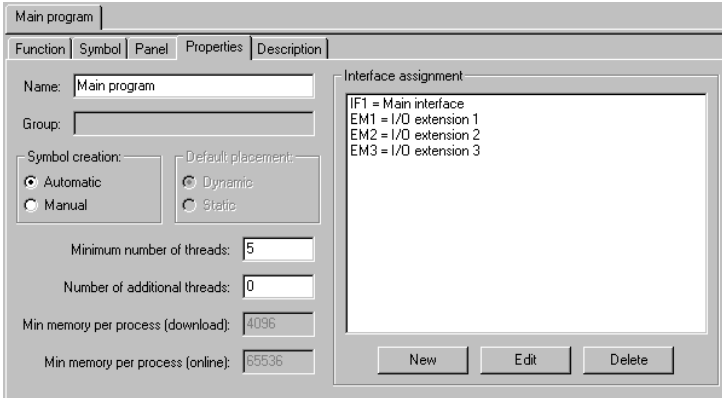
As long as you do not make different settings, you will find the entries **IF1**, **EM1**, **EM2** and **EM3** in the **Interface / Extension** drop-down menu. But you can add to or modify this list. There can be several reasons to do this:

- 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.



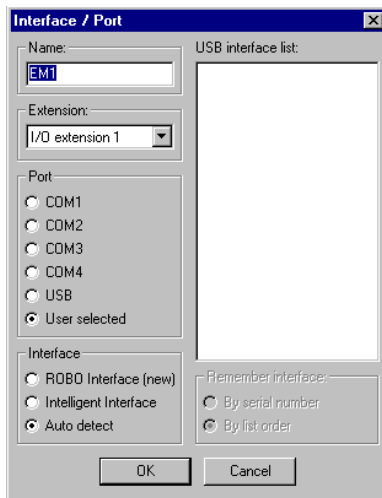
- You might want to run a program, originally written for a ROBO Interface with more than one extension module, by using several Intelligent Interfaces.
- You might want to use more than one ROBO Interface or more than one Intelligent Interface in your program.

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 (Interface or extension) have been assigned to the names **IF1** to **EM3**. 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:

- 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 3.
- 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 Interface with several extension modules, this is the simplest, because this way someone else can use your program unaltered. If you are using several Interfaces, you specify here the port to which the relevant Interface is connected.
- Under **Interface** you can specify whether you wish to use a ROBO Interface or an Intelligent Interface. If the Interface is connected via a serial port, the program can detect automatically which type of Interface is involved (**Automatic** selection).



COM/USB

- The right-hand part of the window is only important if you have connected several ROBO Interfaces to the USB bus. If, under **Port**, you click on **USB**, you can select one of the Interfaces under **USB Interface list**.

Caution: If you would like to operate more than one Interface on the USB bus, you must first assign to each Interface its own serial number. By default, all Interfaces are 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 6.5 *Changing the Interface serial number or firmware version* on page 139.

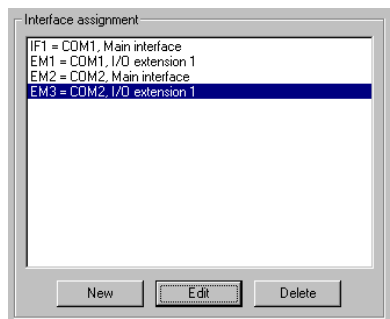
- Under **Remember Interface** you can specify how the program remembers the selected Interface. There are two possibilities here: If you select **By serial number**, the program stores the serial number of the Interface. Even if you connect other Interfaces to the USB bus and remove them, the program 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 number 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.

### 6.3 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.

### 6.4 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.

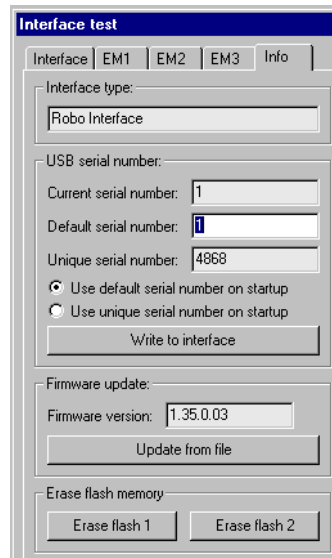


## 6.5 Changing the Interface serial number or firmware version

By default, all ROBO Interfaces and ROBO I/O-Extensions are supplied with the same serial number. As long as you only want to use one 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.
- 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.
- 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 NT, 2000 and XP. 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.

## 7 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.

### 7.1 Basic elements (Level 1)

#### 7.1.1 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.

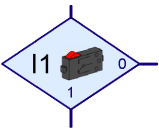
#### 7.1.2 End



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.

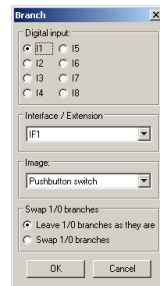
#### 7.1.3 Digital Branch



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.

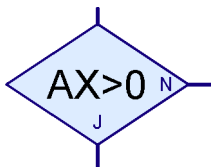
If you right-click on the element, the Properties window is displayed:

- Buttons **I1** to **I8** allow you to enter which of the 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 Chapter6 *Extension modules and controlling several Interfaces* on page 136.
- 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**.

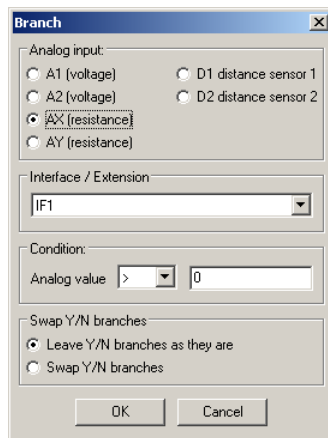
### 7.1.4 Analog Branch



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

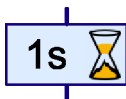
Branch you can compare the value of an analog input with a fixed number and, according to the result of the comparison, branch to the Yes (Y) or No (N) exit.

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



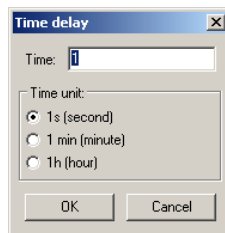
- Under **Analog input**, you can select which of the Interface's inputs is to be queried. All analog inputs return a value between 0 and 1023. You can find further information about the various analog inputs in Section 7.6.2 *Analog input* on page 159.
- Under **Interface / Extension** you can select 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 6 *Extension modules and controlling several Interfaces* on page 136.
- 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**.

### 7.1.5 Time delay



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

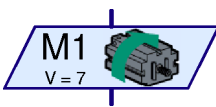
If you right-click on the element the Properties window is displayed. Here you can enter the time delay in seconds, minutes or hours. The time delay can be set over a range from one millisecond (that's one thousandth of a second) to 500 hours (that's just under three weeks). However, the time measurement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

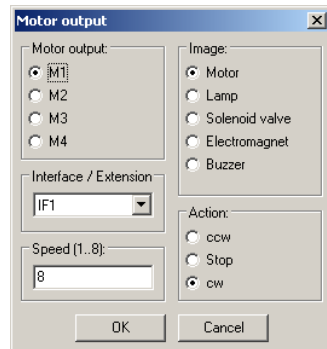
## 7.1.6 Motor output



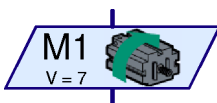
With the program element **Motor output** you can switch one of the Interface's two-pole outputs **M1-M4**. The outputs from the Interface can be used for motors as well as for lamps or electromagnets. With a motor, you would like to be able to set the speed as well as the direction of rotation.

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

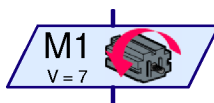
- Under **Motor output** you can set which of the four motor outputs **M1** to **M4** should be used.
- Under **Interface / Extension** you can select whether you want to use an output of the Interface or an output of an extension module or of another Interface. You can find out more about this in Chapter 6 *Extension modules and controlling several Interfaces* on page 136
- 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.



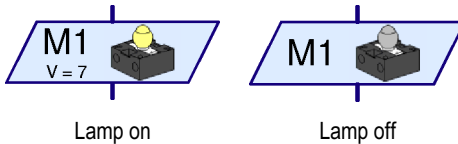
Motor right (cw)



Motor left (ccw)

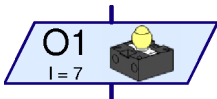


Stop motor



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.

### 7.1.7 Lamp output (Level 2)

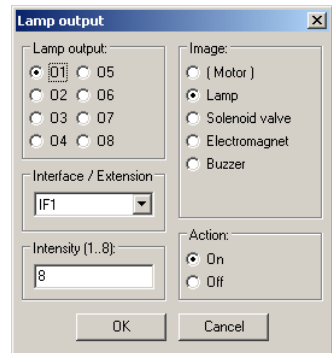


With the **Lamp output** program element you can switch one of the Interface's single-pole outputs **O1-O8**. The Interface's outputs can be used either in pairs as motor outputs (see above) or individually as lamp outputs **O1-O8**. Unlike motor output, lamp outputs only take up one connection pin.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

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 to the ground socket.

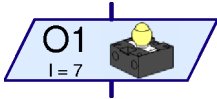
If you right-click on the element, the Properties window is displayed:

- 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 6 *Extension modules and controlling several Interfaces* on page 136.
- 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.

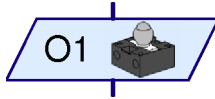




Here are listed the symbols for the various actions for the **Lamp** image.

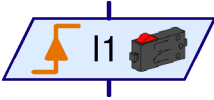


Lamp on



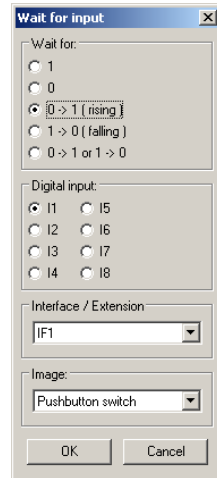
Lamp off

## 7.1.8 Wait for input



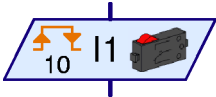
The **Wait for Input** element waits until one of the Interface's inputs is in a particular state or until it changes in a particular way.

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 108 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 6 *Extension modules and controlling several Interfaces* on page 136.
- 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.

### 7.1.9 Pulse counter



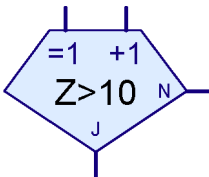
Many fischertechnik model robots also use pulse wheels. These gear wheels operate a sensor four times for every revolution. With these pulse wheels you can turn a motor on for a precisely defined number of revolutions rather than for a given time. To do this, you need to count the number of pulses at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



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 6 *Extension modules and controlling several Interfaces* on page 136
- 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.

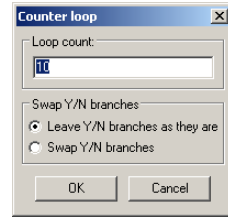
### 7.1.10 Counter loop



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 110.

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.



## 7.2 Subprogram I/O (Level 2-3)

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

### 7.2.1 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 114.

### 7.2.2 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 114.

### 7.2.3 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. If you right-click on the element the Properties window is displayed. There you can give the command input a name, which will then be displayed in the symbol. There is a thorough explanation of the use of this element in Section 5.5: *Command inputs for subprograms* on page 128.

## 7.2.4 Subprogram command output (Level 3)

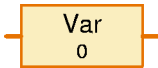


Via this element commands such as left, right, stop can be sent to motors or other output elements in the main program or in the higher-level subprogram. In the subprogram's green symbol that is inserted into the higher-level program, one connecting pin for each Subprogram command input is inserted on 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. If you right-click on the element the Properties window is displayed. There you can give the command output a name, which will then be displayed in the symbol. There is a thorough explanation of this element in Section 5.5: Command inputs for subprograms on page 128.

## 7.3 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.

### 7.3.1 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 7.4.1 = *Assignment*) on page 153). 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 7.6.6 *Panel input* on page 162).

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

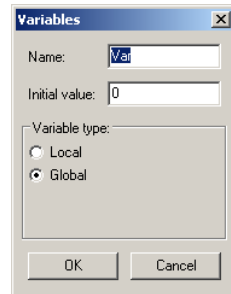
Command	Value	Action
=	-32767 to 32767	Sets the value of the variable to the value passed with the command.

+	-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.

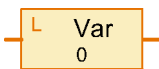
Incidentally, the odd value range of -32767 to 32767 results from computers calculating in the binary system, and not in the decimal system as we do. In the binary system 32767 is a round number, a bit like 9999 in the decimal system. But we don't need to worry about this, as the computer converts all 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.
- The **Variable type** 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.



### 7.3.2 Local variables

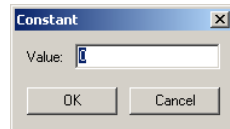


All **global** variable elements with the same name use one and the same variable and always have the same value. That is presumably what you expect and what is generally practical. But if you use variables in subprograms, that can lead to big problems. If your program has more than one parallel process, 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 variables 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.

### 7.3.3 Constant

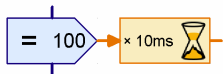
**0** — Like a variable, a constant has a value, but this value cannot be altered by the program. You can link a constant with a data input of a subprogram symbol, if the subprogram is to use the same value at all times. Constants are also very practical for calculations with operators. You will find an example of this at the end of Section 5.7 *Operators* on page 132.



### 7.3.4 Timer variable

**x 10ms**  — 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.



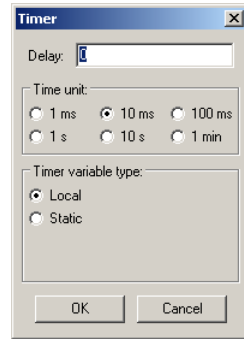
The number of steps to be counted down is generally assigned to the timer via an = command from a command element. In the example illustrated, 100 steps of 10ms each are counted down. This corresponds to a duration of 1000ms=1 second. The precision of this is 10ms.

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 7.3.2 *Local variables* on page 149)






## 7.3.5 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.

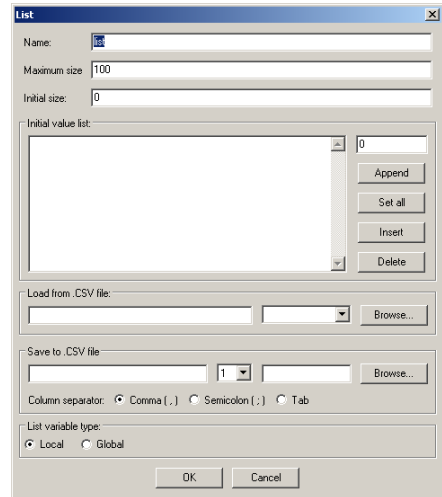
Via the **I** (for Index) input, a specific element of the list can be selected. To do this, you send an = command to the **I** input with the desired element number. The first element is element number 0. Another value can be assigned to the element selected via the **I** input by sending an = command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

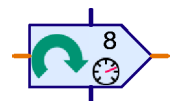
### Properties window for lists

- Under **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 be 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 variable type** you can set whether the list is a global or a local variable (see Section 7.3.2 *Local variables* on page 149). 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.



## 7.4 Commands (Level 3)

All program elements in this group are command elements. Depending on the application they may also be 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 com-

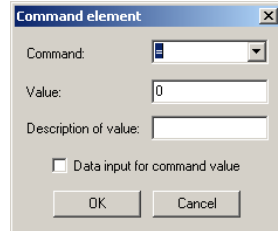




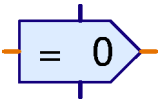
mand or a message to the element connected to the output on its right. There are various commands like right, left or stop, which have different effects on the connected element. As a rule, the connected elements understand only a few commands. The commands each program element understands and the effects 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 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.



#### 7.4.1 = (Assignment)

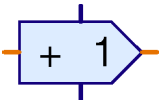


The = command assigns a value to the receiver. As a rule, it is used to assign a value to variables, timer variables, list elements or panel outputs.

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.

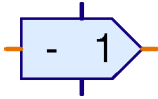
#### 7.4.2 + (Plus)



The + command is sent 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 7.3.1 *Variable* on page 148 and Section 7.3.4 *Timer*

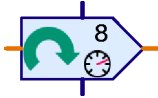
*variable* on page 150.

### 7.4.3 – (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.

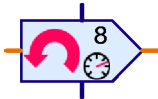
### 7.4.4 Right



The **Right** command is sent to motor output elements to switch on the element with clockwise rotation. See Section 7.6.4 *Motor output* on page 161.

The value is a speed from 1 to 8.

### 7.4.5 Left



The **Left** command is sent to motor output elements to switch the motor on in a counterclockwise direction. See Section 7.6.4 *Motor output* on page 161.

The value is a speed from 1 to 8.

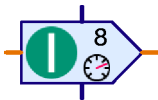
### 7.4.6 Stop



The **Stop** command is sent to a motor output element to stop the motor. See Section 7.6.4 *Motor output* on page 161.

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

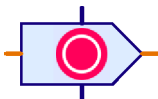
### 7.4.7 On



The **On** command is sent to a lamp output element to switch the lamp on. See Section 7.6.5 *Lamp output* on page 161. 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.

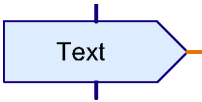
### 7.4.8 Off



The **Off** command is sent to a lamp output element to switch the lamp off. See Section 7.6.5 *Lamp output* on page 161. An **Off** command can also be sent to a motor output element; it corresponds to the **Stop** command.

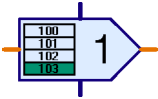
No value is passed with the **Off** command.

### 7.4.9 Text



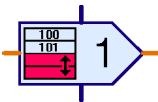
The **Text** command is a special command in that it doesn't send a command with a number, but rather a text of your choice, to the connected element. However, there is only one program element that can process the Text command, and that is a text display in a panel. You will find further information in Section 8.1.2 *Text display* on page 167.

### 7.4.10 Append value



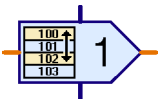
The **Append** command is a special command for list elements. See Section 7.3.5 *Liste* on page 151. The command is accompanied by a value, which is appended to the end of the list. If the list is already full, the command is ignored.

### 7.4.11 Delete value(s)



The **Delete** command is a special command for list elements. See Section 7.3.5 *Liste* on page 151. 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.

### 7.4.12 Exchange values

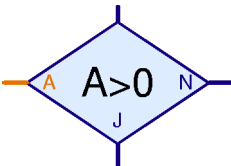


The **Exchange** command is a special command for list elements. See Section 7.3.5 *Liste* on page 151. With this command, any element of a list can be exchanged with the first element. The number of the element to be exchanged with the first element is passed with the command as a value. **Important:** 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.

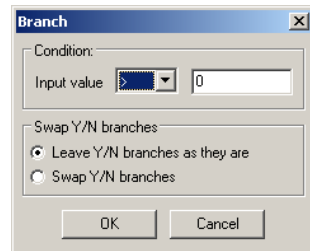
## 7.5 Compare, wait for, ... (Level 3)

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

### 7.5.1 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 7.6.1 to 7.6.6 from page 158). The data input **A** can also be linked to data outputs from variables, timer variables or operators (see Section 7.7 *Operators* on page 163). The element compares the value at the data input



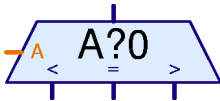
**A** with a fixed, but freely definable value. According to whether the comparison holds or not, the element branches to the **Y** or to the **N** exit.

### 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.
- 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**.

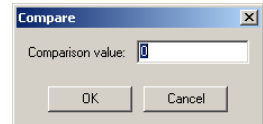
## 7.5.2 Comparison with fixed value



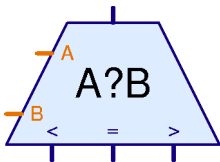
With the program element **Comparison with fixed value**, the value in the data input **A** can be compared with a fixed, but freely definable value. According to the value present at the data input **A** is greater than, less than, or equal to the fixed value, the comparison element branches to the 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.

### Properties window for Compare

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



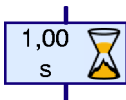
## 7.5.3 Compare



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.

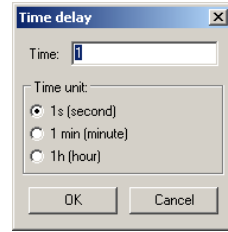
## 7.5.4 Time delay



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 108.

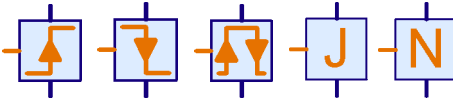
### Properties window for Time delay:

- 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.

### 7.5.5 Wait for...

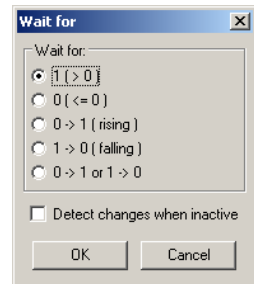


The **Wait for...** program element holds up program execution until a change has taken place or a specific state has been reached in the data input of the element.

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 ( $\leq 0$ ) 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.



### 7.5.6 Pulse counter



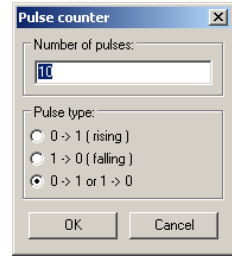
This program element waits for a definable number of pulses at the data input on the left side before it continues program execution. This is very practical for simple positioning tasks with pulse wheels. For

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.

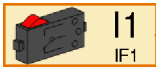
The possibility of recognizing changes when the element is not active, as can be done with the simple **Wait for ...**, is not available for this element.



## 7.6 Interface inputs/outputs

This group of program elements contains all input and output elements. How to use these elements is explained in Chapter 5 *Level 3: Variables, panels & Co* on page 122.

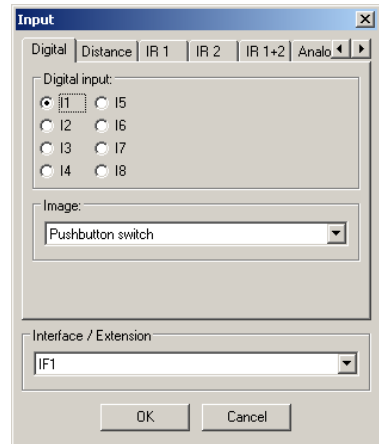
### 7.6.1 Digital input



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 6 *Extension modules and controlling several Interfaces* on page 136.



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.

## 7.6.2 Analog input



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	0-5,5k $\Omega$
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.

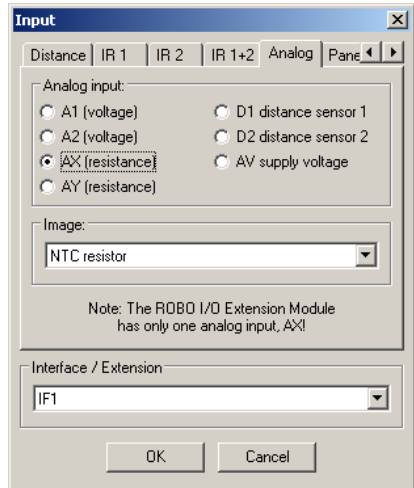
The distance sensor inputs **D1** and **D2** accept connection to special fischertechnik sensors that can measure the distance to, for example, an obstacle.

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 6 *Extension modules and controlling several Interfaces* on page 136.

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.



### 7.6.3 IR Input



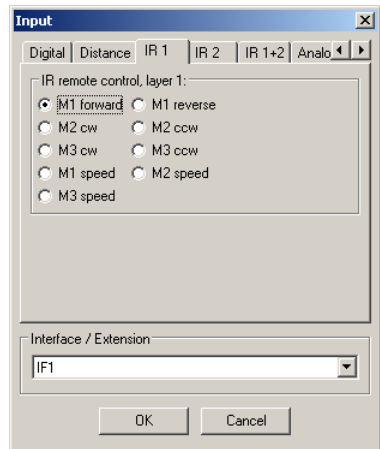
The ROBO Pro Interface has a built-in infrared receiver for the hand-held transmitter from the fischertechnik **IR Control Set**, item number 30344. The infrared hand transmitter is very useful, not only for remote control, but also generally as a keyboard for your models. There are two receivers for 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.





## 7.6.4 Motor output



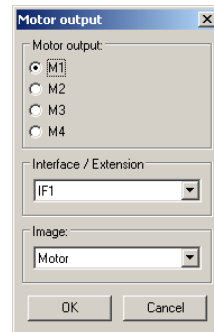
The **Motor output** element allows one of the 4 two-pole motor outputs of a ROBO Interface or an Intelligent Interface to be controlled. A motor output always uses two Interface connections, whereas a lamp output only uses one connection. You can find out more about the difference between motor and lamp outputs in Sections 7.1.6 *Motor output* on page 143 and 7.1.7 *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

### 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 **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 6 *Extension modules and controlling several Interfaces* on page 136.
- 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.



## 7.6.5 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

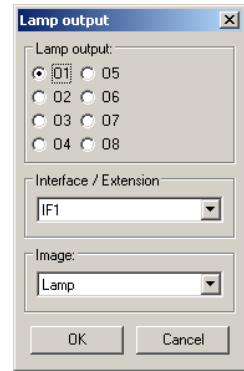
only switch on or off a load device connected in this way; you can't reverse its polarity. You can learn more about the difference in Sections 7.1.6 *Motor output* on page 143 and 7.1.7 *Lamp output (Level 2)*.

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 **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 learn more about this in Chapter 6 *Extension modules and controlling several Interfaces* on page 136.
- Under **Image** you can select an image of the load device connected to the output. In most cases 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.



## 7.6.6 Panel input



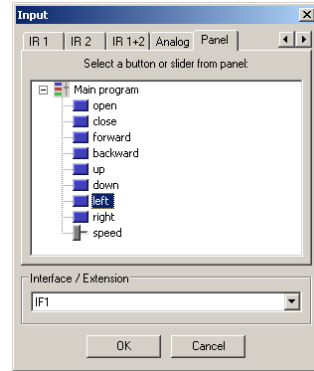
ROBO Pro offers you the possibility of designing your own panels for your models. You can learn more about this in Chapter 8 *Panel elements and panels: overview* on page 166. This makes it convenient for you to control your models from the computer. Push buttons, slider controls and data entry elements 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 110.

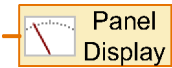
### Properties window for Panel inputs:

One panel is associated with each main program or sub-program. 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.



### 7.6.7 Panel Output



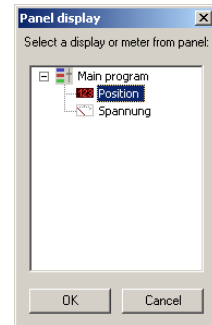
ROBO Pro offers you the possibility of designing your own panels for your models. You can learn more about this in Chapter 8 *Panel elements and panels: overview* on page 166. Alongside push buttons and other input elements to control your model, you can also insert display elements in

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 110.

### 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.



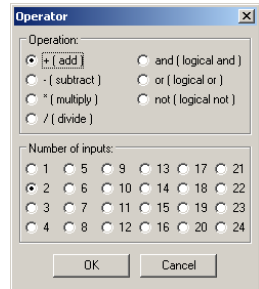
## 7.7 Operators

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.



### 7.7.1 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.

### 7.7.2 Logical operators

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

And	Or	Not
$A > 0 \text{ and } B > 0$	$A > 0 \text{ or } B > 0$	$A \leq 0$

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  $\leq 0$ . 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 Panel elements and panels: overview

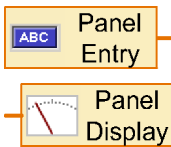
In ROBO Pro you can define your own panels. Panels make it less cumbersome to control complex models. A panel is displayed on your PC screen. Panels only work in online mode. On this subject, see Section 3.7 *Online and download operation-what's the difference?* on page 110.

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.

### 8.1 Displays

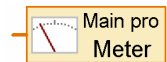
Displays are used in a similar way to Interface outputs. You can set the value of a display with an = command.

#### 8.1.1 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**.

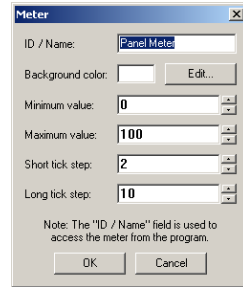


You set the value of the meter by sending an = command to the corresponding panel output in the program. Almost all program elements with data outputs send an = command when their value changes. You can connect analog inputs or variables for example directly to the panel output.



## 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.



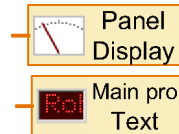
## 8.1.2 Text display



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

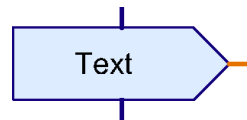
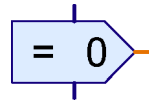
The text display is controlled from the program via a panel output. You will find the **Panel output** in the element group **Inputs, outputs**.

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.



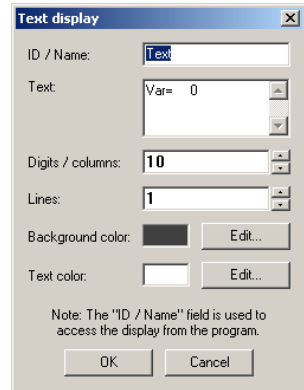
## Control characters in Text commands

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 <b>period</b> as separator.
##,##	Outputs the value in the data input as a number with two decimal places, with a <b>comma</b> 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.



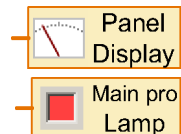
### 8.1.3 Display lamp



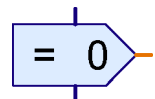
The **Display lamp** is the simplest type of display. It functions in a similar way to a fisher-technik lamp component connected to an Interface output.

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

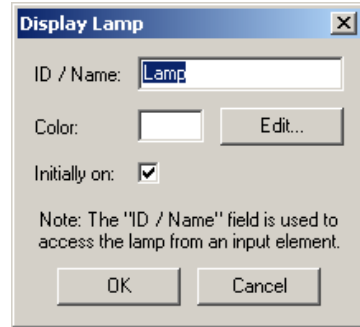




is switched off

### 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.



## 8.2 Control elements

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

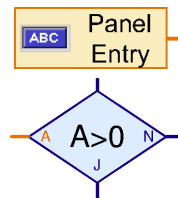
### 8.2.1 Button



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

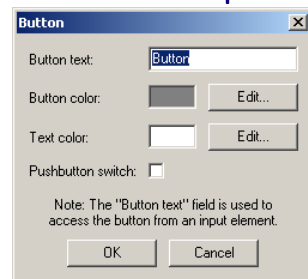
The Button is queried from the program via a **panel input**. You will find the **Panel input** in the element group **Inputs, outputs**.

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.



### 8.2.2 Slider



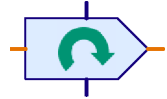
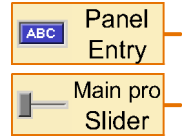
You can use the **Slider** like a potentiometer connected to an analog input of the Interface. Unlike the button the slider can return not only the values 0 and 1, but many different values, like an analog input. The

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**.

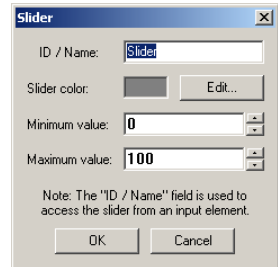
As soon as you have linked the Panel output with a slider by means of its Properties window, the symbol changes and the name of the panel (e.g. Main) and of the slider appear.

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

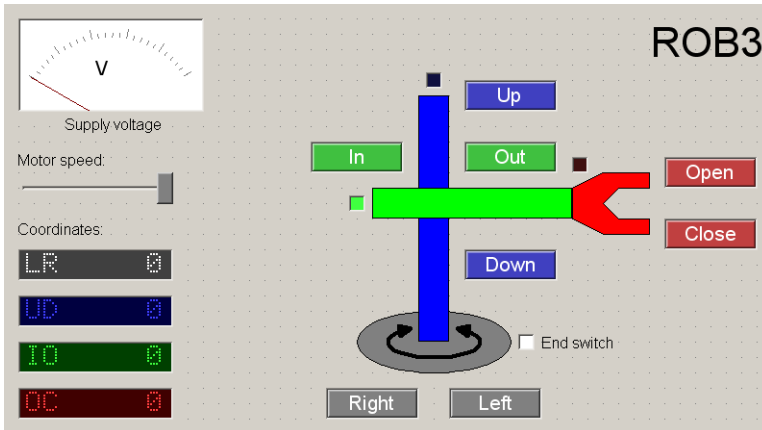
- Under **ID / Name** you should first enter a name for the slider. The name is important so that you can distinguish between more than one slider in your program.
- Under **Slider knob color** you can change the color of the slider knob. To do this, click on **Edit**.
- Under **Minimum value** and **Maximum 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 find them 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 subgroups contain functions to alter color and line thickness.

## 9 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.