FX-RP multiDISPLAY getting started manual

Details of the FX-RP multiDISPLAY

Physical

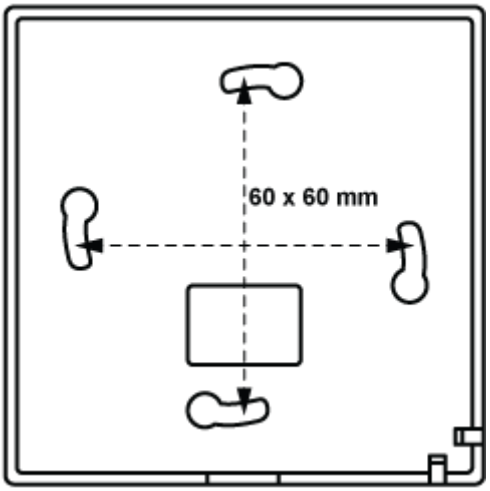
The Fidelix FX-RP multiDISPLAY is a 3.5” room panel (usable surface: 70x53mm). The FX-RP-A has a straight enclosure, the FX-RP-B is slightly tilted. The display can also be ordered without enclosing, for embedding in electrical cabinets, or other third-party enclosures.

FX-RP-A

FX-RP-B

FX-RP-C

The multiDISPLAY encasing has 4 holes in the back to screw it onto a standard pattress box.



The supported operating temperature is between 0 and +50°C.

IP-class is 20.

The multiDISPLAY has an on-board NTC10 temperature sensor.

Nominally, the multiDISPLAY is powered with 24 VDC, but power supply voltages from 12-45 VDC, or 16-32 VAC are supported.

Power consumption is ≈1W at 100%, or ≈0.5W at 40% display brightness.

Technical

The Fidelix FX-RP multiDISPLAY is a serial Modbus RTU over RS485 device that can communicate with speeds up to 115 200 bps. It can be used as Modbus slave (addresses 1..255), or Modbus master to one or several slaves.

The multiDISPLAY has an internal clock and a capacitor that saves the time and date in case of a power interruption.

The multiDISPLAY has 5 independent internal time schedules.

The multiDISPLAY as Modbus slave has 250 built-in 32-bit datapoints, which can be read or written to using 2 read- and 2 write-registers per datapoint. As Modbus master, the multiDISPLAY can be configured to read/write registers on a limited number of slaves. Typically though, it is used to visualise only one slave-controller.

The multiDISPLAY has 320x240 pixels on its 3.5” screen.

Symbols or images can be any of the standard graphics file formats. Files encoded with more than 256 colours cannot be compressed by the converter, and will thus take up a lot more memory space. The converter program will notify the memory used by your project at the end of the conversion process.

Software needed for programming the FX-RP multiDISPLAY

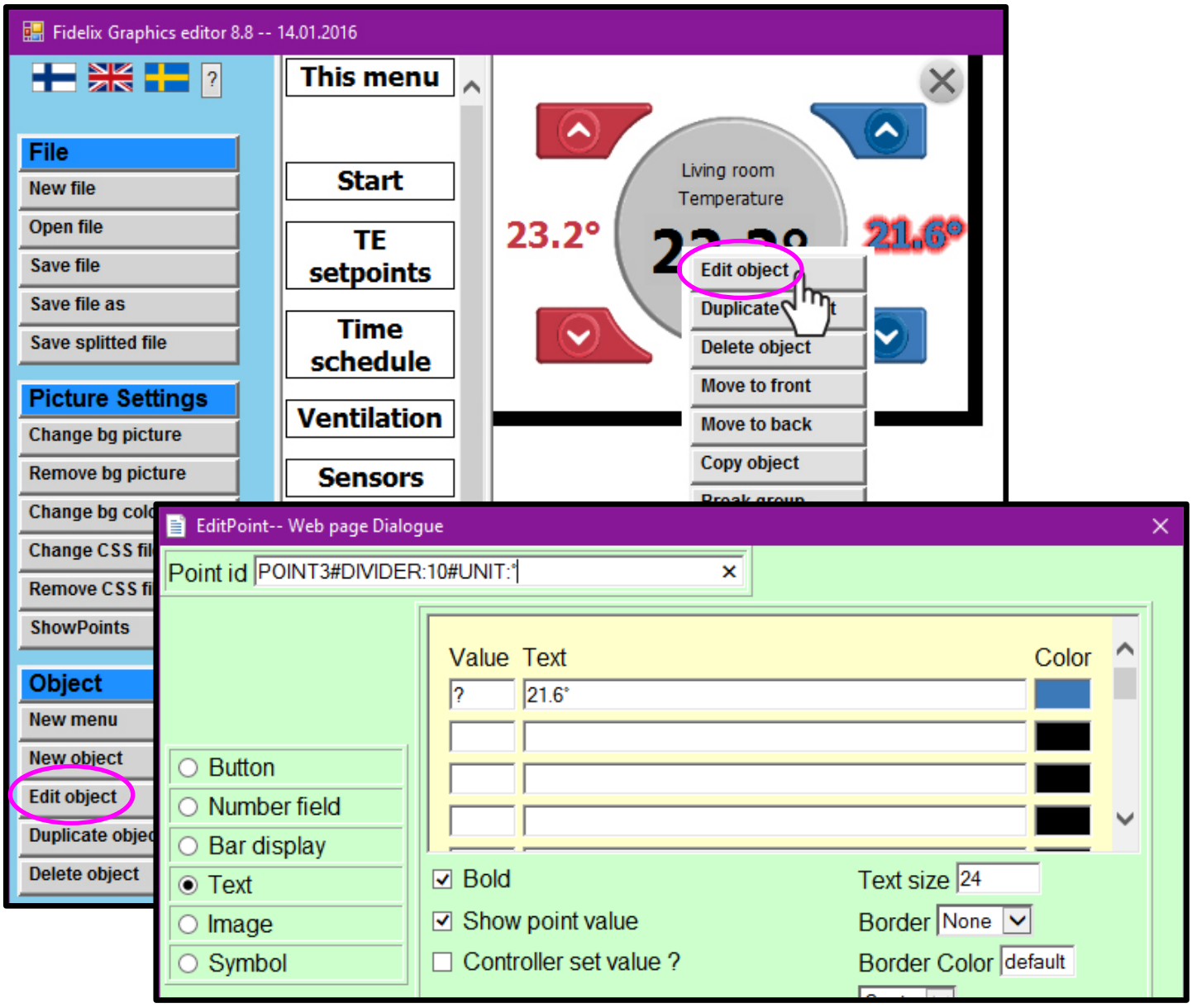
Fidelix Graphics editor / HTML-editor

The Fidelix graphics editor is used to create HTML pages. From the top-left corner, 320 by 240 pixels can be used to make “pages” for the multiDISPLAY.

Each page can contain (register) values, images, background graphics, “active symbols” (a different colour image for each possible register value e.g. ON/OFF = ● / ● to show device feedback status), “buttons” (a different image for each possible register value e.g. AUTO/MANUAL = A / ⟳ to show device desired control status), links to other pages etc.

In slave mode, each register value is identified as a “POINTxx” (POINT1..POINT250).
In master mode, the slave register is defined in the same field following this syntax:
“MODBUS:ADD=1:REG=0:TYPE=HOLDING:FORMAT=INT16”

These parameters are set in the “EditPoint” popup window you get when right-clicking a selected object (a selected object is outlined in red) and choosing “Edit object” or by selecting an object and subsequently clicking the “Edit object” button in the menu in the left panel:

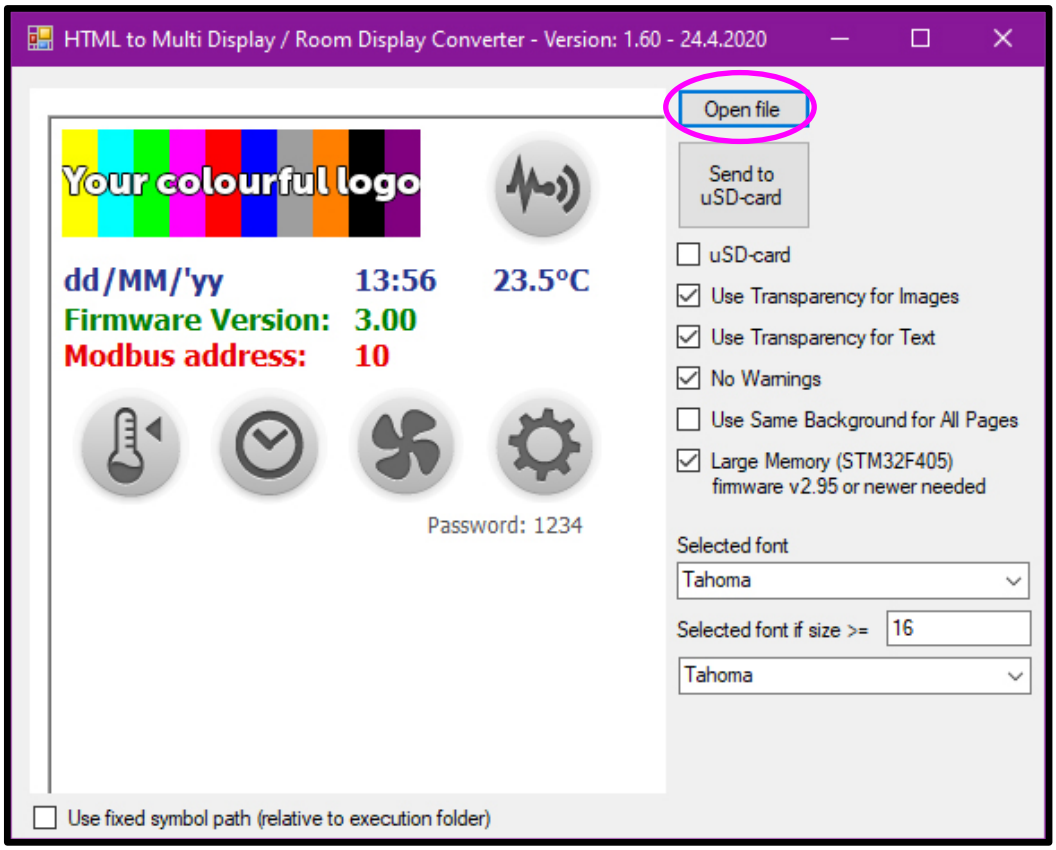


The FdxIndex.htm file is loaded into the middle column of the graphics editor. This file is not visible on the multiDISPLAY, but helps navigation between different pages during the programming stage.

A short introduction video about how to use the Fidelix graphics editor / HTML editor can be found here: youtu.be/2VugdB8Vhzo.

HTML to display converter

Once you are satisfied with the way your pages look on your computer, start the converter program. This program converts the HTML pages to the multiDISPLAY format; simply click the “Open file” button and open the home page of your project. The converter will follow all links it finds, take screenshots of each page to create page backgrounds, and generate the necessary files for the multiDISPLAY.



Basics

Project folder setup

Typically, a project's graphics should reside in one folder. In that folder are:

- The HTML pages that will become the different screens on the multiDISPLAY
- A folder called "Symbols" with all used symbols inside of it (see later)
- When working in Modbus master mode, the "Modbusmastersettings.txt" file with Modbus communication parameters
- When working with non-Latin characters, the "CharacterSet.txt" file containing all characters you wish to use in your project.
- When odd or even parity is needed to communicate with the Modbus master, the "serial.txt" file containing a single line "Parity=Even" or "Parity=Odd"
- The "FdxIndex" file, making the navigation between pages more comfortable during programming.

Demo project

To make learning easy, a demo project is provided with the download of the graphics editor. It will be used as reference further through this manual.

The display as a slave has 250 datapoints; POINT1..POINT250. Each datapoint is linked to predefined Modbus registers.

Open the Graphics editor / HTML editor and open the "Setpoint.htm" page inside the "multiDISPLAY demo project" folder. Double click on the red "23.2°" value to open the "EditPoint" window (the same window can be opened by selecting "Edit object" from the right-click menu, or by selecting the value and clicking the "Edit object" button in the menu on the left).

The PointID reads "POINT2#DIVIDER:10#UNIT:°". A PointID always starts with "POINTxx". All additional parameters are appended at the end in random order, separated by the "#" symbol. "DIVIDER:10" means that the register value will be shown with one decimal.

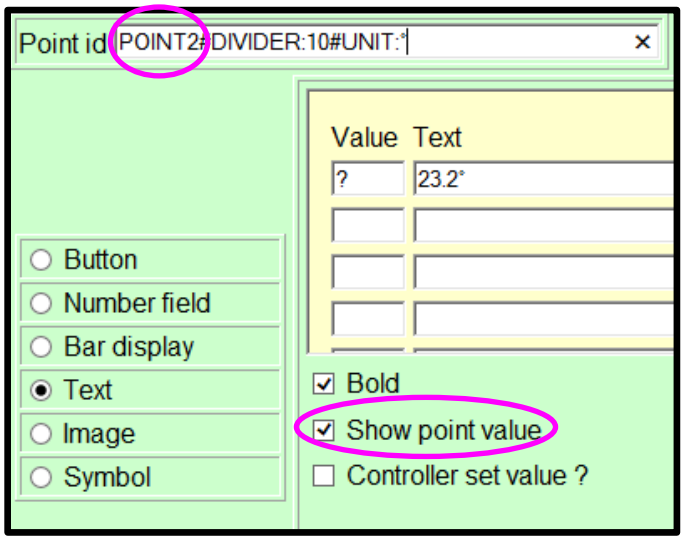
Load the FdxIndex file into the middle column by clicking the "Load" button in the bottom of the column and from there, easily open the Modbus master page. Open the EditPoint window for any of the value fields to see that the PointID for a display that is configured to operate as Modbus master has a syntax like this: "MODBUS:ADD=10:REG=1005:TYPE=HOLDING:FORMAT=UINT16".

For Modbus master register values as well, additional parameters can be given for the visualisation of the value by appending the parameters at the end of the PointID.
e.g. "#DIVIDER:10#MIN:180#MAX:250#UNIT:°C#UNITSPACE"

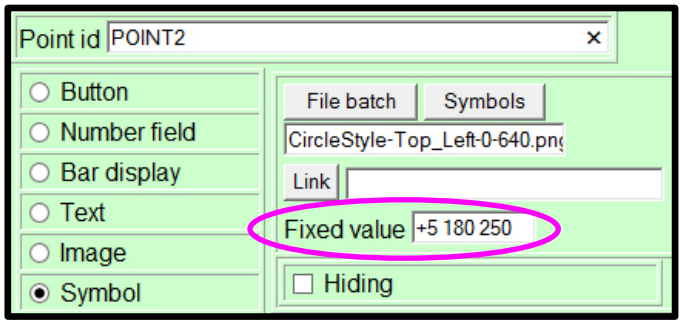
Showing/changing register values with visible numeric value

Go back to the setpoints page and open the PointEdit window for one of the setpoint values (image right). Having “Show point value” ticked, means we show the value of the register on the multiDISPLAY.

The text in the yellow part of the window (here:23.2°) is a placeholder to facilitate designing your pages in the graphics editor, but has no significance as to what is shown on the actual multiDISPLAY, if the unit and/or divider is defined in the suffixes to the PointID.



Open the EditPoint window for one of the two red arrow symbols (up or down buttons) and you can notice the same pointID (“POINT2”) is used, now in combination with the “Fixed value” field:



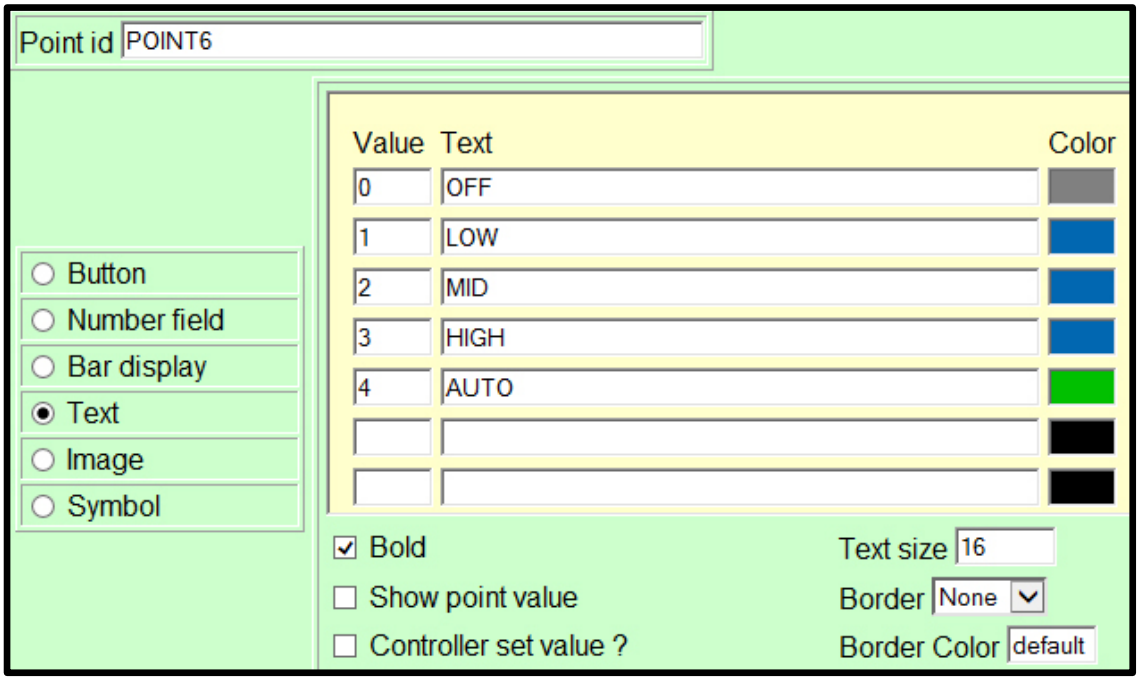
The three values “+5 180 250” here are respectively the change of value applied when pressing the element (+5), the minimum (180), and the maximum value (250) set to the register value.

This means that when pressing the “up” or “down” button on the screen, the setpoint value will increase/decrease in steps of 5 from 180 to 250.

The suffix “#DIVIDER:10” in the PointID field of the setpoint values is applied to that visualisation only. This means that the register will contain for instance “215” and this will be shown as “21.5°” because the suffix is “ #DIVIDER:10#UNIT:° ”. This is also why the “+5” in the fixed value field, is shown as steps of 0.5.

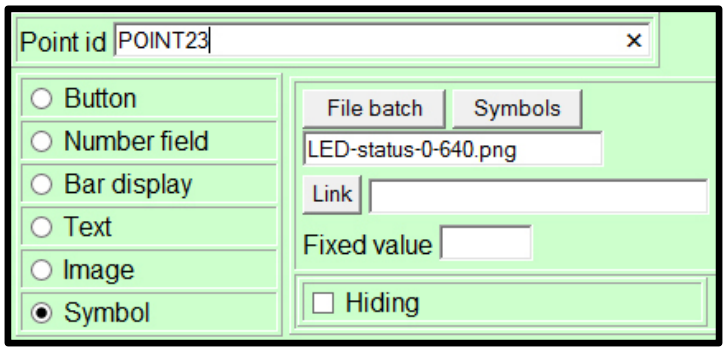
Showing registers values with text

Open the ventilation page and open the EditPoint window for POINT6, which reads “OFF” in the graphics page. You will see the “Show point value” box is not ticked, and for each possible register value, the corresponding text and colour is specified in the yellow part of the window:

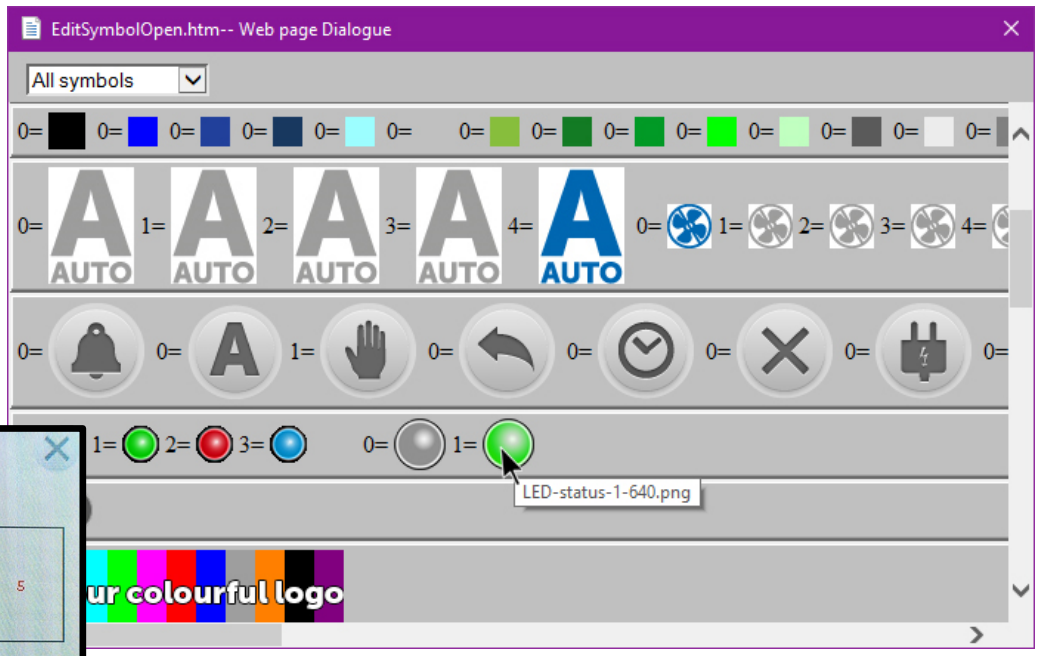


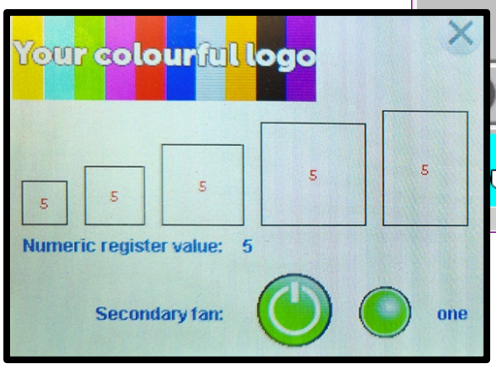
Showing registers values with symbols

On the same ventilation page, you will also find the indication “LED” for the secondary fan “POINT23”.



Click on the “Symbols” button in the EditPoint window to see all available symbols in your project. You will see that all symbols have a file for value zero, and some also have a symbol for other values. The symbol shown on the display, corresponds with the register value.

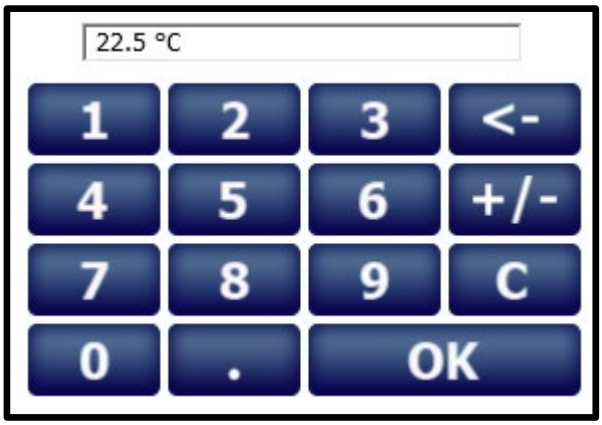




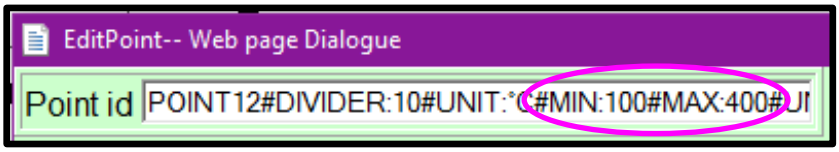
If the register has a value for which no symbol is available, a black square with the value in red is shown (see image on the left):

Changing register values with user-input values

Open the Sensor values page and open the EditPoint window of one of the setpoints. Both “Show point value” and “Controller set value” are ticked. This means that when the value is clicked, a popup window will ask user input:



Since the user can now enter any value they like, it is wise to set a minimum and maximum accepted value. These limits are set in the suffix of the PointID:

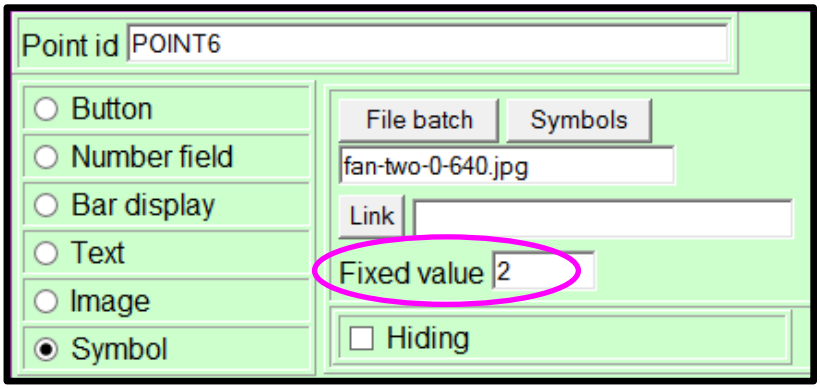


Changing register values with user-changeable symbols

Another, third way to allow users to change register values (aside direct input and with +/- buttons with fixed value increments), is by assigning values directly to symbols:

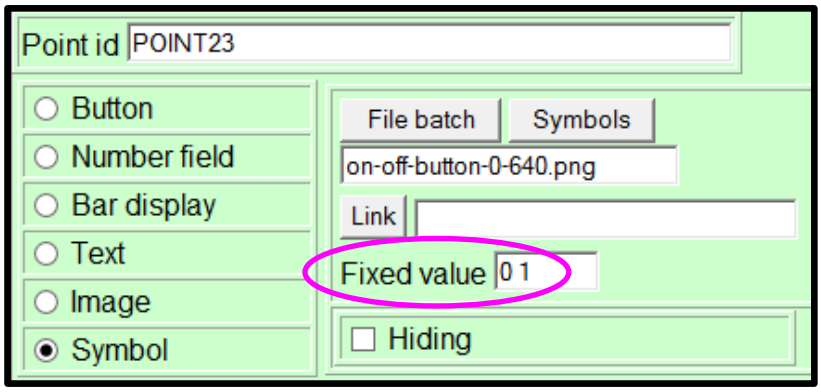
Open the ventilation page and open the EditPoint window for one of the fan images.

The “Fixed value” field only has one value. Clicking on the image will set the register to that value.



Open the EditPoint window for the secondary fan control button:

This button will toggle the register value between all values in the “Fixed value” field. In the example, only zero and one are possibilities, but more values can be summed up in this field.



Internal multiDISPLAY registers

The multiDISPLAY has a set of internal values that are also stored in registers. Examples are the temperature measured by the on-board NTC10 sensor in various formats, the internal time schedules’ statuses, the number of the point of which the value was last changed, …

These registers do not offer all options that normal “POINTS” offer, but can be visualised by using “REGxxxx” in the PointID field.

An example of how to use these register values can be found in the Time schedule page.

More details about and a full list of the internal registers can be found in the full programming manual.

Create your own project for the FX-RP multiDISPLAY

Modbus communication parameters

multiDISPLAY is Modbus slave

The multiDISPLAY will automatically detect the bus speed from 4800 to 115200 bps.
Also the number of databits and stop bits are automatically detected from the bus.

The default parity is set to “no parity”. To use even or odd parity, place a file called “serial.txt” in your project folder, containing a single line “Parity=Odd” or “Parity=Even” (or “Parity=No”).

Each of the 250 datapoints now has 6 registers linked: 0..750 are used to receive data from the Modbus master, and 1000..1750 are used to set local user input to the internal registers. More details about the setup of these registers can be found in the multiDISPLAY programming manual, or you can watch this 4-and-a-half-minute video on Youtube to clarify the register setup: https://youtu.be/tEushV8ugsA.

multiDISPLAY is Modbus master

The registers numbers and types are specified in the graphics, but the bus setup needs to be defined in the “ModbusMasterSettings.txt” file you need to put in your project folder when the multiDISPLAY is Modbus master.

At the very minimum, this file needs to contain this:

```
BAUD=57600
BITS=8n1
SENDDELAY=250
TIMEOUT=250
MODBUSDEVICES=AUTO
COMBINE=50
```

There are a lot more possibilities for setup and even the possibility to send readings from one slave to another. You can also send “internal registers” from the display itself (Internal multiDISPLAY registers) to the connected slave(s). This and more is explained in the multiDISPLAY programming manual.

Symbols

Inside each multiDISPLAY project, a folder called “Symbols” has to be present. This folder contains the symbols used in your project.

The syntax for naming the files is “YourFileName-x-640.jpg” where “x” represents the register (POINT) value. The image for the “zero” value needs to be present always, as this is used in the graphics editor.

Files can be bmp, jpg, jpeg, gif (non-dynamic), or png. Transparency can be used and transparent parts can overlap.

Ideally, to use the smallest amount of memory, you create symbols that can be used in their original size, but symbols can also be resized in the EditPoint window’s “Width” and “Height” fields.

Images that have no PointID (“Unknown”) will be considered as part of the background image that is created for each page in the conversion process.

Start with an existing page

Copy the FdxIndex.htm page and any other page from the demo project to your own project folder. This will allow you to have the borders and menu for each multiDISPLAY project.

Start editing, duplicate pages, rename, link and relink, and add your own datapoints as needed. If you forget to remove links, the converter program will complain about this and abort the conversion.

Convert and upload

Convert HTML to .bin

Open de HTML to multiDISPLAY program, make sure to enable or activate the applicable settings (mostly, transparency is wished for and you will do a conversion for a display with large memory (the V1 display with smaller chip was discontinued in 2017), and after a few conversion, you will possibly no longer want or need the basic warning messages), and start the conversion by clicking the “Open file” button and selecting what will become your project’s homepage.

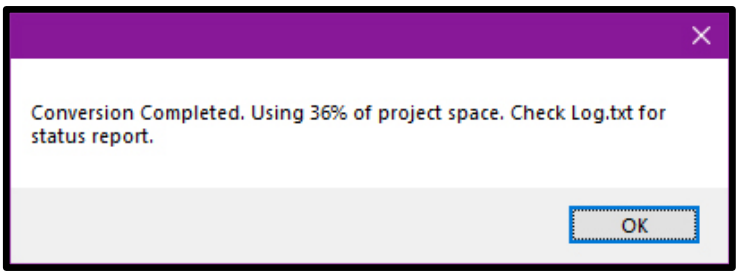
Keep the “uSD-card” option unticked. Selecting this option stops the project from being loaded into the memory of the display, and will make the display read data from the memory card continuously. This is a useful feature during the designing phase, but should never be used for actual projects.

The converter will follow all links it encounters, open those pages, take screenshots to generate background images, convert the images, and compress those that it can.

All these files will be created inside a folder called “UserFiles”, which will be created inside the folder where the converter program is situated.

Once the converter is ready, it will let you know and you can proceed to copy the created “UserFiles” folder to the root of your µSD memory card.

You can do this manually, or use the “Send to uSD-card” button. The converter will then search for the first available drive that has a “UserFiles” folder and overwrite it.



Load your project into the multiDISPLAY’s memory

Put the memory card in the slot marked with “SD ⇓”, and power up the multiDISPLAY.

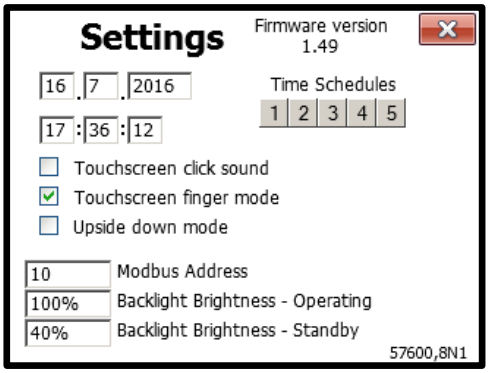
Once the “Loading data, please wait!” message disappears, the multiDISPLAY will show the page you selected in the converter. At this point, the display is ready. You can verify correct communication is happening by checking the RxD and TxD LEDs on the PCB of the display. RxD blinks when a Modbus message is read from the bus, TxD lights up when the multiDISPLAY is sending a message on the bus.

You can now power off the display and remove the memory card. When you power up the display again without the memory card in the slot, the project will be loaded from the internal memory.

Set time and date, Modbus address, …

If you didn’t put a link to the settings or the calibration of the display in your project, and didn’t disable the access to these pages, you can access the multiDISPLAY settings page by long-pressing the display in one place for about 10 seconds. The calibration process will be triggered, after which you will see the settings page.

Here you can set time and date, the Modbus address (ignored when you are using the display in Modbus master mode), display brightness, flip the display when it is mounted upside down, and access the 5 internal time schedules (you can create your own custom page for this also; see the “Time schedule” example page in the demo project).



Videos

On the Fidelix Youtube channel, you can find a few short instruction videos that may be useful to consult. Among others, there is a programming introduction video (youtu.be/xW5f6Je9wK4), covering about the same things this manual explains, an introduction to the Fidelix graphics editor (youtu.be/2VugdB8Vhzo), and an in-depth explanation about the multiDISPLAY’s in- and output registers (youtu.be/tEushV8ugsA), teaching how to read-write values between the multiDISPLAY and the Modbus master controller it is connected to.