Intelligent display Communication protocol doc

PB 21.5.22

1) PC-DISPLAY communication

PC-display Communication happens over UART at 115200 baud.

Commands and newlines are terminated by the CR character (\r, carriage return, sent on pressing enter in Putty, ascii 13).

Some examples are available in utils repository https://github.com/PVT-2022-Intelligent-display/python-utils.

Commands

Following commands are implemented at time of writing.

Command	Description
config	Configures all screens and objects on all screens, saving the settings to external memory. More info below
bitmap	Adds a bitmap to external memory. More info below.
delete bitmaps	Deletes structure storing info about bitmaps saved in memory, making them unaccessible and allowing them to be overwritten.
report screens	Prints out information about all screens and all objects curently stored in external memory.
report bitmaps	Prints out information about bitmaps stored in external memory (how many there are, their resolutions).
hex report bitmaps	Same as report bitmaps, but also prints out first 128 pixels of each bitmap.

Bitmap command:

Is used to save a new bitmap into external memory. User first sends the command bitmap followed by CR. Next, pixel resolution in x axis is sent as decimal number in plaintext, followed by CR. Next, pixel resolution in y axis is sent, same as x resolution.

After this, the values of individual pixels are sent. Pixels are represented by 2-byte integers in the RGB565 format. They are sent as 4-char long hexadecimal strings (without the 0x prefix). Individual bytes are not separated by CR or any other character. The last pixel is followed by CR.

Pixels are send in row order, starting from upper left corner.

After all bytes are sent, the display should reply with „[cl] Bitmap #X has been written.“. Note the value of X, you will use it later to refer to this bitmap you have just saved.

Example 1: To save a 3x2 bitmap representing the polish flag, PC needs to send following text over uart:

(note: red in rgb565 is 0xF800, white is 0xFFFF)

```
bitmap\r
3\r
2\r
FFFFFFFFFFFF800FF800FF800\r
```

In the utils repository, you can find a python script that can convert a bitmap from rgb888 to rgb565 and send it into the display. See file bitmapLoader.py

Transparency: A single color value (at time of writing, 0x0001) is reserved to for transparent colour. When converting a bitmap for use by the display, make sure that you don't accidentally create transparent pixels where you don't want them! See above python file for example of implementation.

Note on max resolution: The maximum size of bitmap that can be drawn to the display may be limited (the bitmap has to fit into a buffer in stm32). See objectVisualization.c for up-to-date limitations (constants MAX_BITMAP_DRAW_XSIZE and MAX_BITMAP_DRAW_YSIZE, at time of writing 128x128). If you need to draw a bitmap that is larger than this, split it into multiple bitmaps and place them next to each other.

Note on scaling: If you want to print something that will be 300x300 but doesn't have to be super-detailed, you can use scaling. Objects that make use of bitmaps support scaling them up (eg. If source bitmap is 50x10 and object has scaling = 3, it will be drawn to display as 150x30). See below.

Config command:

Is used to set up screens and objects on display. For an example, see screenInitDemo.py.

The config command can be split roughly into 3 levels:

- General (number of screens)
    - Screen (number of objects on screen)
        - Objects (on current screen)

Command usage works like this:

Step 1: First, pc sends the string config, followed by CR. Then, number of screens to be configured is sent, as decimal number in plaintext, followed by CR.

Step 2: After this, pc sends the string "screen", followed by CR. After that, it sends the number of objects on current screen, as decimal number in plaintext, followed by CR. Screens are configured starting from 0 and going up to (number of screens – 1).

Step 3: After this, the pc needs to send data for all objects on current screen. After all objects on current screen are sent, if any screens remain, go back to step 2.

Object configuration

Configuring an object consists of 2 parts – sending the object header and sending object hex data. The header has the same structure for all objects, while the hex data structure depends on object type. The object header is sent first:

Object header:

Following data is sent, with each line being followed by a CR. All numbers in header are sent as decimal in plaintext.

Name	*Explanation*
Object Type	String identifying what kind of object it is. Eg. "button", "picture"
objectId	Unique identifier.
Xstart	x-coordinate of upper left corner of object
Ystart	y-coordinate of upper left corner of object
Xend	x-coordinate of lower right corner of object
Yend	y-coordinate of lower right corner of object
Datalen	How many bytes of hex data will follow.

Hex Data:

Hex data provides additional information specific to the object (eg. Strings for labels, text offset for buttons…). Each byte of hex data is sent as 2-char hexadecimal string (without the leading 0x). The amount and meaning of hex data depends on object type. You should always send as many bytes as the number you have specified in datalen line of object header.

The last byte of hex data should be followed by CR, but individual bytes are not separated by CR or any other character. When byte number is spans 2 bytes, it signifies that the value is read as uint16_t, with most significant byte first. So for example, hex data of picture using bitmap number 2 at scaling 1 should be 00020001

Object type	**rectangle**
Valid datalens	2

Byte number	Name	Meaning
0-1	Color	Rgb565 color with which the rectangle should be filled

Object type	**picture**
Valid datalens	4

Byte number	Name	Meaning
0-1	Bitmap number	Number of bitmap which will be loaded from external memory and used to draw this object.
2-3	Scaling	How much should the source bitmap be scaled up when drawn to display. When scaling is 1, a 10x5 bitmap will take up 10x5 pixels on display. When scaling is 8, it will take up 80x40 pixels.

Object type	**label**
Valid datalens	8 + 1 per character

Byte number	Name	Meaning
0	Pixelscaling	How much should the 5x8 font be upscaled before being drawn to display.
1	Hspacing	Horizontal spacing. Number of lcd pixels separating two characters in label. Is not affected by pixelscaling.
2	Vspacing	Vertical spacing. Number of lcd pixels separating characters vertically (only relevant if your label contains a newline)
3	Usebg	When 0, font has transparent background. When not 0, background of font is filled with bgcolor
4-5	Textcolor	Rgb565 color to be used for drawing text of this label
6-7	Bgcolor	If usebg != 0, this colour will be used to fill background
8+	Label string	Characters of label string, one byte per character, using extended ascii. Null-termination is not required.

Button and screenbutton objects have variable datalens depending on whether they are printed with a string or just as a bitmap.

If you don't want to print a string over the button, only provide the two bitmap numbers and scaling. (and set datalen to 6 in header!)

Object type	**button**
Valid datalens	Either 6 or 13 + 1 per string char

Byte number	Name	Meaning
0-1	Bitmap unpressed	Number of bitmap used to represent the button in it's unpressed state
2-3	Bitmap pressed	Number of bitmap used when button is pressed
4-5	Scaling	How much should the source bitmap scaled up. See picture for explanation.
6	Xoffset	Offset of string start in x-axis (measured from upper left corner)
7	Yoffset	Offset of string start In y-axis
8	Pixelscaling	How much should font be scaled up. See label for explanation.
9	Hspace	See label for explanation.
10	Vspace	
11-12	textColor	Color of text as rgb565. (note: Background color is never used in buttons and screenbutton)
13+	Text string	See label for explanation.

Screenbutton is the same as button, except that it starts with a different byte and following bytes are shifted as a result

Object type	**screenbutton**
Valid datalens	Either 7 or 14 + 1 per string char

Byte number	Name	Meaning
0	Target screen	Number of screen to which the display should switch when this button is pressed.
1-2	Bitmap unpressed	Number of bitmap used to represent the button in it's unpressed state
3-4	Bitmap pressed	Number of bitmap used when button is pressed
5-6	Scaling	How much should the source bitmap scaled up. See picture for explanation.
7	Xoffset	Offset of string start in x-axis (measured from upper left corner)
8	Yoffset	Offset of string start In y-axis
9	Pixelscaling	How much should font be scaled up. See label for explanation.
10	Hspace	See label for explanation
11	Vspace	
12-13	textColor	Color of text as rgb565. (note: Background color is never used in buttons and screenbutton)
14+	Text string	See label for explanation.

Object type	**slider**
Valid datalens	2

Byte number	Name	Meaning
0-1	Bitmap number	Number of bitmap to be used to visualize this slider.

Interactivelabel object is similar to label object, except it carries no string data with it. Instead, text is only drawn to this object when it is received from external processor.

Object type	**interactivelabel**
Valid datalens	8

Byte number	Name	Meaning
0	Pixelscaling	How much should the 5x8 font be upscaled before being drawn to display.
1	Hspacing	Horizontal spacing. Number of lcd pixels separating two characters in label. Is not affected by pixelscaling.
2	Vspacing	Vertical spacing. Number of lcd pixels separating characters vertically (only relevant if your label contains a newline)
3	Usebg	When 0, font has transparent background. When not 0, background of font is filled with bgcolor
4-5	Textcolor	Rgb565 color to be used for drawing text of this label
6-7	Bgcolor	If usebg != 0, this colour will be used to fill background

2) External processor – display communication

Communication with the external processor happens over the second uart. It is significantly simpler than PC-display communication.

Firstly, the display reports to the external processor whenever an object (like a button or a slider) is pressed. The message is text-based, carrying first the type of object that was pressed and secondly its ID. For example: “Type:2 ID:4”.

The type information corresponds to the integer representing the object type in question, as defined in configStructs.h (where objectType_t is defined). For example, type 2 corresponds to object type button.

Secondly, the display receives messages from external processor. Currently, those messages are only intended to print text to objects of type interactivelabel. The parsing of those messages is implemented in extProcComm.c. The structure of the message is: “sXXX string\r”. The first character is always ‘s’. XXX is the id of target interactivelabel object, in decimal. This is followed by a space. After this space, all following characters are interpreted as the string which will be printed into said interactivelabel. \r is carriage return.

For example, to print text „hello“ to interactive label with id 12, the message would be: „s12 hello\r“.