Connecting Arduino to Nextion – My Way

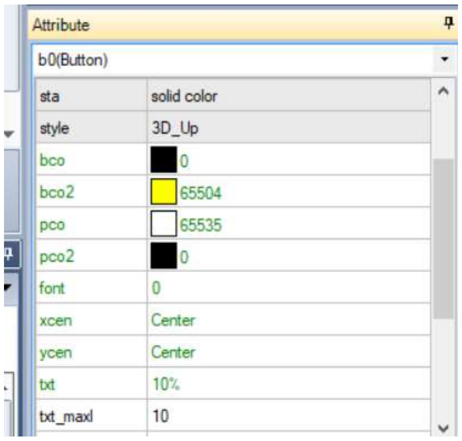
The following is a short tutorial on how I accomplish communication to and from the Nextion. I use these methods for two reasons, its simple, and it doesn't hog the memory of small processors. The latter is quite important if you are dealing with a UNO instead of a Mega. This is why I do not use the Itead library. It's huge and quite frankly I believe overcomplicated.

First let be clear, this is my way and has been adopted from the work of others. It is sometimes not pretty, does not use the feedback codes of the Nextion to indicate whether the message went through or not, but it works, and it works well by being simple.

Arduino to Nextion

What you need to understand is that the Nextion requires you as the programmer to 'command' an item to change if you want it changed. That's it. And that is what I believe the itead Arduino library designers missed as a simple programming paradigm.

Commands are as simple as those found in the Nextion Instruction Set Commands i.e Click, vis etc (https://nextion.itead.cc/resources/documents/instruction-set/) or by simply commanding any one of the "Green" items that show up in the Attributes of any one of the toolbox items (buttons etc) to change to a new value.



Attribute

b0(Button)

sta	solid color
style	3D_Up
bco	■ 0
bco2	🟨 65504
pco	□ 65535
pco2	■ 0
font	0
xcen	Center
ycen	Center
txt	10%
txt_maxl	10

And if we look hard at all of the building blocks the Nextion IDE provides (Buttons, Text, Variables, Numbers etc) then you quickly realise the there is only two other item types that you change, either values, or text

So my complete interface for the Arduino to control the Nextion consists of the following four procedures. Yes that's right only four, and if fact one can be deleted.

void NextionInit(int pageId)

This code simply sets the Nextion onto the page that you want it to be on. I usually only use this in setup so ensuring my Nextion goes to the required start up screen. The pageId is the Nextion page number
Example would be

```
        NextionInit(3);                                  // Forces Nextion to go to page 3
```

In fact this could be eliminated altogether as it is a simple Nextion page command.

void NextionSetValue(String Component, int value)

This code sets an integer into any component attribute that requires an integer, so it could write to a 'Number', or to a 'Number Variable'

```
int Max_Val= 34;
NextionSetValue("Operate.Top",Max_Val);
```

In this case I have sent 34 to a Variable Number called ‘Top’ on page ‘Operate’.   I don’t use this style of addressing often as I usually just sent the values or commands when I am on the correct page. So if I know I am on the Operate page then I would just send NextionSetValue (“Top”, Max_Val);
I’ll expand on this a little later on.

void NextionSetText(String component, String txt)
Similar to the SetValue this code sets the text of component.
Example:
NextionSetText ("Operate.t12","My text");
As above text box t12 is on the Operate page and I have set the text to “My Text”

void NextionSendCommand(const char* cmd)
This is the last of subroutines and needs a bit of explaining. Nextion does not accept strings, but a row of characters that form the command. So the correct way of sending a command using this subroutine would be

```
String Command = "vis t1,1";                    // set t1 visible
NextionSendCommand(Command.c_str());
```

The .c_str() is a C++ modifier that converts the string to an array of characters, essentially dropping off the ‘/0’ on the back of the string. However, I have found that Nextion will accept:

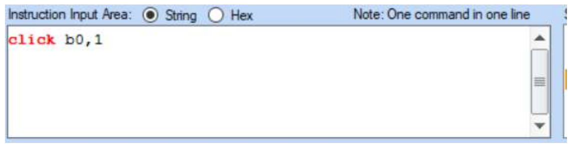
NextionSendCommand ("vis t1,1");

Which is of course much easier to code.

The SendCommand is the most useful code of all. You just write the same string as you would in the ‘Instruction Input Area’ under debug. For example

- NextionSendCommand("click b0,1") ;
- Nextion Send Command(“xpic 5,28,230,50,5,28,2”);
- NextionSendCommand("vis t9,1");

And that’s how to test that the string you are sending as a command actually works. Just copy and paste it into the ‘Instruction input Area’ under debug. If it works there then it will work from the Arduino.


Instruction Input Area: ● String ○ Hex
Note: One command in one line

```
click b0,1
```

Serial Connection

This is where my code is not pretty. As you will realise the above procedures require communication via the serial link to the Nextion. With a Uno you only have one dedicated serial port, so you may have to use Software Serial (though I must admit I have had issues with this on a Uno) Or if you are using a Mega then you can use any one of three hardware serial ports. So let’s say you elect to use Serial2. Then you have to change all of the ‘Serial’s to ‘Serial2’ in the NextionSetValue() and NextionSendCommand() procedures. No big deal. You only need to do it once when you have decided which port connects to the Nextion. With Software Serial you will create a nextion serial port object and this will be put in place of ‘Serial’.

Procedures

So here they are!

```
//---------------------------------------------
// Initiates a start to page selected
//---------------------------------------------
```

```
void NextionInit(int pageId)
 {
 String page = "page " + String(pageId);      //Page
 NextionSendCommand(page.c_str());
 }//end nextion_init


// -------------------------------------------
// Sets an integer value to Number or Variable
// --------------------------------------------
void NextionSetValue(String Component, int value)
  {
  String compValue=Component+".val="+value;
  Serial.print(compValue);
  Serial.write(0xFF);
  Serial.write(0xFF);
  Serial.write(0xFF);
  }

//-----------------------------------------------------
// Sends text
// ----------------------------------------------------
void NextionSetText(String component, String txt)
 {
 String componentText = component + ".txt=\"" + txt + "\"";
 NextionSendCommand(componentText.c_str());
 }

//-----------------------------------------------
//  Sends a command
//-----------------------------------------------
void NextionSendCommand(const char* cmd)
 {
 Serial.print(cmd);
 Serial.write(0xFF);
 Serial.write(0xFF);
 Serial.write(0xFF);
 }
```

Nextion to Arduino

Again this is my way, quite simply I ignore the return codes from the Nextion and don’t use them. Also I like to have a clear idea of what is coming down from the Nextion for debug and clarity purposes. So I use the Nextion ‘print’ statement lots and I order the items sending information back in such a way that I know exactly where the information comes from. This makes it a lot easier to use in my code whether in the Nextion or in the Arduino.

Pages

Usually you need to know what page you are on. So if I am in Page 3 then I put

```
print “P3”
```

into the ‘Postinitialize Event’ of that page. So my Arduino receives “P3” when the page is entered and refreshed. I only do this for pages I need to communicate with.

Buttons

With buttons I always order them by objname on the page they are on. So if I had four buttons on Page 3 then there objnames would be B31, B32, B33, and B34. On the Touch Press or Touch Release Event of each button I simply put a print statement with the button identifier i.e.

```
    Print “B31”
```

So again, on receiving this in the Arduino I know exactly which button on which page was pushed.

This is important as say you have two buttons b0 on pages 3 and 4. How do you know what was pushed? You can use the fact that you know the page number but my method is easier. (Also if you send a command “click b0,1” then both buttons would operate depending on what page you are on)

I use another similar technique if I am using the buttons as a selection. Say I have four buttons and each is a selection of hours. So button 1 allows the user to select one hour, button 2 two hours etc.

In this case, I would have the buttons print “H1”, “H2”, “H3”, and “H4” when pushed.

Arduino Serial Reception

Using the Nextion print statements allows me to use standard serial reception methods in the Arduino. I then combine this with switch statements to determine the action.

So if I set up an incoming message array msg[5] then I can use something like the following to have the Arduino do something as each page is entered.

```
// responds to page changes

 If (msg[0]==”P”){

switch (msg[1])

          {

          case ‘1’:   ……………………

          case ‘2’:    ……………………..

          case ‘3’” ………………………

          }

}
```

Obviously the same idea works for buttons, and the hour values as above. Simple eh!

With the Buttons ordered by page and item number (B34) then you have two pieces of information in one transmission, the page and the button serial number. I must admit I do not use the page reference this way, I always use the P3 idea as this allows me to order my code as shown above. i.e I know where my page code is and I know where my button code is.

The second advantage to carefully defining what is coming down from the Nextion is that you have essentially predetermined the initial character send. For the examples above the first character must be a ‘P’ or a ‘B’ or an ‘H’. This means that you can reject any message that does not have these three characters as the starting character.

Knowing the Page

I find this very handy for two reasons. Firstly I can set up code to print to the Nextion only when I am on that page. This limits the transmission and also allows me some options. For example I can separate out the slow and fast actions I may need. So under the ‘slow’ page procedure I may set items visible, but another procedure may rapidly print values for a quick update to the same page.

As I showed above when I am sure of the page I am on I can send “t23”,”My text” rather than “Operate.t23”, “My text”

Finish

I don’t like the Itead library, but that is not to say that it may be more to your liking. I have written some very sophisticated programs for Arduino/Nextion using only the above four procedures without any discernible issues and without any loss of functionality.

I am certain that techniques suggested will make it easier for beginners to obtain a quick, structured and understandable connection to the Nextion.

--oo--