Nextion Instruction Set

From ITEAD Wiki

Contents

- 1 Note:
- 2 Classification I: Operation Commands of Component and System
  - 2.1 page: Refresh page
  - 2.2 ref: Refresh component
  - 2.3 click: Activate component’s press/release event
  - 2.4 ref_stop: Stop refreshing screen
  - 2.5 ref_star: Recover refreshing screen
  - 2.6 get: Get variable/constant value with format
  - 2.7 sendme: send current pageID to UART
  - 2.8 cov: variable type conversion
  - 2.9 touch_j: Touch calibration
  - 2.10 vis: Hide/show component
  - 2.11 tsw: Enable/disable component touch function
  - 2.12 com_stop: Stop execute instructions from UART
  - 2.13 com_star: Execute instructions from UART
  - 2.14 randset: Set random value range
  - 2.15 code_c: Clear the buffer
  - 2.16 print: Get variable variable/constant value without format
  - 2.17 printh: Send hexadecimal string
  - 2.18 add: Add data to waveform component
  - 2.19 addt: Add data to waveform component in volume
  - 2.20 cle: Clear data in waveform component
  - 2.21 rest: Reset Nextion device
  - 2.22 doevents: Force refresh the screen immediately
  - 2.23 strlen: check length of a string
  - 2.24 if: if statement
  - 2.25 while: while statement
  - 2.26 for: for statement
  - 2.27 repo: read EEPROM - Enhanced Model Only
  - 2.28 wepo: write EEPROM - Enhanced Model Only
  - 2.29 wept: write data in hex to EEPROM through UART - Enhanced Model Only
  - 2.30 rept: read data in hex from EEPROM to UART - Enhanced Model Only
  - 2.31 cfgpio: configure GPIO - Enhanced Model Only
- 3 Classification II: GUI Designing Command
  - 3.1 cls: clear the screen
  - 3.2 pic: display picutres
  - 3.3 picq: Crop picture
  - 3.4 xpic: Advanced crop picture
  - 3.5 xstr: Print string on the device
  - 3.6 fill x, y, w, h, color
  - 3.7 line x, y, x2, y2, color
  - 3.8 draw x, y, x2, y2, color
- 3.9 cir x, y, r, color
- 3.10 cirs x, y, r, color

4 Nextion HMI: System Variables List
5 Nextion HMI: Color Code List
6 Format of Device Return Data
7 Useful link

Note:

1. The instruction is end with three bytes "0xff 0xff 0xff"
2. All the instructions and parameters are in ASCII
3. All the instructions are in lowercase letters

Classification I: Operation Commands of Component and System

page: Refresh page

```
page pageid
```

pageid: Page ID or Page Name

- Example 1:

```
page 0  //Refresh page 0
```

- Example 2:

```
page main  //Refresh the page main
```

Remarks:
The device automatically refresh page 0 when power on.

ref: Refresh component

```
ref cmpID
```

cmpID: component ID or component name

- Example

```
ref t0  //refresh component t0
ref 1  //refresh component 1
```
Remarks:

1. The default loading mode is automatically load when you create and edit a component in Nextion Editor. If set it as manually load, you should use ref command to load the component. Or when the component is covered by the other components, you can use this command to refresh the covered component.

2. Component auto refresh when attribute changes only valid for those attributes in green bold font. The other attributes not in this format can only be refreshed and displayed by the command of ref.

3. When creating a component in Nextion editor, the default loading mode is auto loading. When the loading mode is set as manual loading, it requires to use ref to load. When the component is sheltered by GUI command mapping, or when the component is sheltered by other manual loading components, you can use ref to refresh the component.

---

**click: Activate component's press/release event**

`click cmpID, event`

cmpID: component ID or component name

event: 0 - release event; 1 - press event

- Example

```
click b0, 1   // activate press event of component b0
click 2, 0    // activate release event of component 2
```

Remark:

1. Use "click" command to activate press/release event of a component without touch the LCD.

**ref_stop: Stop refreshing screen**

- Example

```
ref_stop // no parameters
```

Remarks:

1. If you do not want to see the whole waveform flow progress, rather, you want to view the whole waveform all at once, you can stop refreshing the screen and refresh it when all adopted points being passed through.

2. Once stop refreshing the screen, all other commands and responded attribute assign operations will still run normally, but the components on the screen will not auto-refresh, neither will any changes in component attributes be refreshed and displayed on the screen (the changes are valid). When the
device receive the command ref_star, those changes will be immediately refreshed and displayed on the screen (on condition that the changes are made from the attributes in green bold font).

3. When stop refreshing the screen, ref command will still be executed. Besides, all GUI drawing instructions such as draw point, draw line will still be executed and any changes will be immediately displayed.

---

**ref_star: Recover refreshing screen**

ref_star: recover refreshing screen

- **Example**

```c
ref_star  // no parameters
```

Remarks:

This command should be used after ref_stop command.

---

**get: Get variable/constant value with format**

get att

att: variable name

- **Example 1**

```c
get t0.txt  //return t0's txt value
```

- **Example 2**

```c
get j0.val  //return J0's val value
```

- **Example 3**

```c
get "123"  //return constant string "123"
```

- **Example 4**

```c
get 123     //return constant value: 123
```

Remarks:

1. When returned value is a string, the returned data is 0X70+ASCII code+0xff 0xff 0xff.

2. When returned value is numerical, returned data is 0X71+4 byte binary data+0xff 0xff 0xff. The data storage mode is little-endian mode (namely, low-order in front, and high-order at back).
3. The specific returning format of data, please refer to the table: Format of Device Return Data

---

**sendme: send current pageID to UART**

- **Example:**

```
sendme //no parameters
```

**Remarks:**

If you want the page auto-send pageID every refresh, simply enter sendme in page initialize event.

---

**cov: variable type conversion**

cov att1,att2,lenth

att1: source variable

att2: target variable

lenth: length of the string (0 - automatic length, nonzero - fixed length)

- **Example 1:**

```
cov h0.val,t0.txt,0 //convert the value variable of slider h0 val into decimal string and assign the txt variable of t0, the length is automatic
```

- **Example 2:**

```
cov t0.txt,j0.val,0 //convert the string variable of t0 txt into value and assign the variable of slider h0 val, the length is automatic
```

**Remarks:**

1. lenth always represents the length of the string, when the value converts into string, it is the length of target variable; when the string converts into value, it is the length of source variable.

2. If the target variable and source variable are of the same type, the conversion failed.

---

**touch_j: Touch calibration**

touch_j:

- **Example:**

```
touch_j //Enter touch calibration function, this command does not need parameter
```

**Remarks:**
All the devices have been calibrated before packing from the factory, this command is not needed under normal circumstances

**vis: Hide/show component**

`vis obj,state`

*obj: component name or component ID*

- **Example 1:**
  
  `vis b0,0 //hide component b0`

- **Example 2:**
  
  `vis b0,1 //show component b0`

- **Example 3:**
  
  `vis 1,0 //hide the component whose ID is 1`

- **Example 4:**
  
  `vis 1,1 //show the component whose ID is 1`

**Remarks:**

1. The first parameter 255 means all components in current page, for example: `vis 255,0` (hide all components in current page); `vis 255,1` (show all components in current page).

**tsw: Enable/disable component touch function**

`tsw cmp,state`

*cmp: component name or component ID*

*state:* (0=disable or 1=enable)

- **Example 1:**
  
  `tsw b0,0 //component b0 touch invalid`

- **Example 2:**
  
  `tsw b0,1 //component b0 touch valid`
Example 3:

```
tsw 1,0  //component of ID 1 touch invalid
```

Example 4:

```
tsw 1,1  //component of ID 1 touch valid
```

Remarks:

1. The first parameter 255 means all components in current page, for example: tsw 255, 0 (all components in current page touch invalid); tsw 255, 1 (all components in current page touch valid).

**com_stop: Stop execute instructions from UART**

Example:

```
com_stop  //no parameters
```

Remarks:

This command is used for pausing the execution of serial port commands, but note that the device will continue receiving the commands and store them in the buffer. Until receiving "com_star" commands, the device will execute the rest commands that store in the buffer.

When using this command to pause the execution, please make sure whether the buffer size and the maximum capacity of command queue can store all the commands you need. You will find these two parameter in Nextion Hardware manual.

**com_star: Execute instructions from UART**

Example:

```
com_star  //no parameters
```

Remarks:

After receiving this command, the device will execute all the commands that store in buffer and new commands form UART.

When using com_stop and com_start, please make sure whether the buffer size and the maximum capacity of command queue can store all the commands you need. You will find these two parameter in Nextion Hardware manual.

**randset: Set random value range**
randset minval,maxval

minval: minimum value

maxval: maximum value

- Example:

```plaintext
randset 1,100 //set current value randomly generated from 1 to 100
```

Remarks:

1. You should use randset to set random value generated range beforehand. Without setting randset, the value range will be 0~4294967295 by default. After setting randset, you will get a new random value within the preset range every time you run rand.

2. The range set by randset keeps valid unless the device being reboot or reset.

---

**code_c: Clear the buffer**

- Example:

```plaintext
code_c //no parameters
```

Remarks: Clear the commands that is stored in the buffer without execution.

---

**print: Get variable variable/constant value without format**

print att

att: variable name

- Example 1:

```plaintext
print t0.txt //returns the txt property value of component t0 in ASCII
```

- Example 2:

```plaintext
print j0.val //returns val's property value of component j0 in 4-byte hexadecimal data
```

- Example 3:

```plaintext
print "123" //returns the ASCII of string "123":0x31 0x32 0x33
```

- Example 4:

```plaintext
print 123 //returns the 4-byte hexadecimal data of value "123":0x7B 0x00 0x00 0x00
```
Remarks:

1. When the variable obtained by using print command is string type, the device directly returns string ASCII; if it is numeric type (such as progress val property), the device will directly return the variable’s 4-byte hexadecimal data, value is stored as little-endian mode (ie low level in the front, high level at the back).

2. When use print command to obtain data, the device sends only the data content, no start identifier, nor end mark.

3. print command can match up printh command to add a piece of user-defined label in the front so as to tell the microcontroller which component this variable belongs to.

4. print command is very similar to get command, the only difference between them is get command returns data with initial identifier (0x70 or 0x71) and end mark (0xff 0xff 0xff), while print not.

**printh: Send hexadecimal string**

printh hex hex: the characters' hexadecimal string expression to be sent

- Example

```plaintext
printh d0 a0 //let device send these two bytes 0xd0 0xa0
```

Remarks:

1. print and printh commands are executed only in Nextion display, they will not be shown in the software simulator.

2. When using printh command to send data, the device sends specified characters only, no Start character, Space character or End character.

3. There must have one and only Space separated between each set of characters in the parameter, both upper case and lower case are supported in the hexadecimal string expression.

**add: Add data to waveform component**

add objid, ch, val

objid: Waveform component ID

ch: Waveform component channel number

val: value (maximum 255, minimum 0)

- Example 1

```plaintext
add 1, 0, 30 //add data 30 to channel 0 of the Waveform component which ID number is 1
```
Example 2

| addt, 1, 50  //add data 50 to channel 1 of the Waveform component which ID number is 1 |

Remarks:

1. Waveform component only support 8-bit values, 0 minimum, 255 maximum.

2. Each page supports up to four Waveform components, each Waveform component supports up to four channels. It supports continuously pass through data, the component will auto-flow and display the value. It supports to change attributes during passing through data, such as change the background color or foreground color for each channel during the process.

**addt: Add data to waveform component in volume**

addt objid, ch, qty

| objid: waveform component ID |
| ch: channel number of waveform component |
| qty: adopted point quantity of the data |

Example:

| addt, 1, 0, 100  //waveform component whose ID is 1 enter into data pass through mode, pass through adopted point quantity is 100 |

Remarks:

1. Waveform component only support 8-bit values, 0 minimum, 255 maximum. Single pass through is 1024 bits maximum.

2. After sending waveform data pass through command, it will take some time to get the device responded and started passing through values. This period takes about 5ms(it will take longer if there are other commands to be executed in the buffer zone before executing data pass through command). Following this period, it will send a data pass through ready data(0XFE+Terminator) to user, then it will start sending pass through data. The data being passed are only hexadecimal numbers. It recovers to command receiving state only after the device has adopted specified data.

3. Waveform will not refresh until specified data has been passed through completely.

**cle: Clear data in waveform component**

cle objid, ch

| objid: waveform component ID (component name or variable is not supported); ch: channel |

Example:
cle 1, 0 //clear the data of waveform component ID=1, channel=0;
cle 1, 255 //clear all the data of waveform component ID=1

Remarks: ch=255: all channels

**rest: Reset Nextion device**

rest

Example:

```plaintext
rest //no parameters
```

**doevents: Force refresh the screen immediately**

doevents

Example:

```plaintext
doevents //no parameters
while(n0.val<1024)
{
    n0.val++
doevents //the screen will keep refreshing during the loop
}
```

Remark: 1, In a long looping statement, you can use doevents to avoid the screen display like brick. 2, doevents usually work with "while" or "for" statement.

**strlen: check length of a string**

strlen att0, att1

att0: source string variable att1: return value, length of att0

Example:

```plaintext
strlen t0.txt, n0.val //assign length of t0.txt to n0.val
```

Remark:

1, att0 must be string, att1 must be numeric.

**if: if statement**

- Example 1: If to.txt equals to 123456, it will switch to page 1
if(t0.txt=="123456")
{
Page 1
}

- Example 2: Below codes can switch content of txt of button component b0 between start and stop in press event)

```javascript
if(b0.txt=="start"
{
  b0.txt=="stop"
} else
{
  b0.txt=="start"
}
```

- Example 3: Below codes can switch content of txt of button component b0 among 1, 2, 3 in press event)

```javascript
if(b0.txt=="1"
{
  b0.txt=="2"
} else if (b0.txt=="2"
{
  b0.txt=="3"
} else
{
  b0.txt=="1"
}
```

Remarks:

1. Numerical variable supports operators: “>, <, ==, !, >=, <=”
2. Character and String variable supports operators: “==, !=”
3. Nested “()” operator is not allowed. It does not support comparison operation, such as: if (j0.val + 1> 0).
4. Operator “()” must be paired.
5. Supports nested “if” and “else if” statement.
6. Complex expression, such as “if (j0.val+1>0)” is not supported
7. Download "if" HMI demo here: Demo (http://support.iteadstudio.com/helpdesk/attachments/1029054663)

**while: while statemet**

Example:

```javascript
//n0.val will auto-add to 100, in this process the screen will
//not refresh the result until all the statements of this process
//being completely executed.
while(n0.val<100)
```
 Remarks: 1, In the process of executing a while loop, the device will not show corresponding touch event, serial command will be received in buffer, but not executed until all the statements in the process has finished execution. Please be cautious when using this command, or it might be easily enter into endless loop. 2, It's recommended to use "doevents" command together. 3, refer to remark of "if" statement

**for: for statement**

- Examples:

```
//n0.val will auto-add to 100, in this process the screen will
//not refresh the result until all the statements of this process
//being completely executed.

for(n0.val=0; n0.val<100; n0.val++)
{
}

//n0.val will auto-add to 100, in this process the screen will
//keep refreshing and displaying the result of component n0.

for(n0.val=0; n0.val<100; n0.val++)
{
doevents
}
```

**repo: read EEPROM - Enhanced Model Only**

repo att, add

att: variable/constant

add: storage address in EEPROM

Example:

```
repo t0.txt,10 // read and assign value to t0.txt, read length is (t0.txt-ml)+1
repo n0.val,10 //read and assign value to n0.val, read length is 4 bytes
```

Remark:

1. read a string from EEPROM, read length is length of the string+1.
2, read a numeric value from EEPROM, read length is 4 bytes.

---

**wepo: write EEPROM - Enhanced Model Only**

wepo att,add

att: variable/constant

add: storage address in EEPROM

Example:

```plaintext
wepo t0.txt,10 // write value of t0.txt to EEPROM, start from address #10. storage size is (t0.txt-mal)+1 bytes
wepo "abcd",10 // write string "abcd" to EEPROM, start from address #10. storage size is 5 bytes
wepo 11,10 // write constant 11 to EEPROM, start from address #10. storage size 4 bytes
```

Remark:

1, Storage address start from 0byte to 1023byte.

2, Storage size of a string is 1 byte + string length; storage size of a numeric constant is 4 bytes.

3, Supported numeric data type is long integer, from -2,147,483,648 to 2,147,483,647 [v035 supports 0 to 2,147,483,647 only]

---

**wept: write data in hex to EEPROM through UART - Enhanced Model Only**

wept add, length

add: start address in EEPROM

length: length of data

Example:

```plaintext
wept 30, 20 // write 20 bytes of hex data from UART to EEPROM, start address is 30
```

Remark:

1, When wept instruction received, Nextion device will take around 5ms to initiate and send ready signal "0xFE 0xFF 0xFF 0xFF"

2, Before you send data from UART, you should wait until you get ready signal from Nextion device

3, wept can not be terminated.

---

https://www.itead.cc/wiki/Nextion_Instruction_Set
**rept: read data in hex from EEPROM to UART - Enhanced Model Only**

rept add, lengt

add: start address in EEPROM

length: length of data

Example:

```
rept 30, 20 // read 20 bytes of hex data from EEPROM to UART, start address is 30
```

**cfgpio: configure GPIO - Enhanced Model Only**

cfgpio id, state, cmp

id: I/O pin number

state: work state of the GPIO

0 - pull up input mode
1 - input binding mode
2 - push pull output mode
3 - PWM output mode
4 - open drain output mode

cmp: binding component, available in work state 2

Example:

```
cfgpio 0, 0, 0    //Configure GPIO0 as pull up input mode, you can read the input status from system variable pio0,
                  //such as: n0.val = pio0

cfgpio 1, 2, 0    //Configure GPIO1 as push pull output mode, you can control the output level
                  //by system variable pio1, such as: pio1 = 1

cfgpio 2, 1, b0    //Configure GPIO2 as input binding mode, the binding component is button b0.
                   //Falling edge of GPIO2 will trigger b0's button press event, rising edge of
                   //GPIO2 will trigger b0's button release event.

cfgpio 4, 3, 0    //Configure GPIO4 as PWM output mode, you should use system variable pwm4
                  //to set duty cycle before you use the instruction
```

Remark:

1, only GPIO4 to GPIO7 support PWM function

2, When configure a GPIO to input binding mode, only components in current page can be bind. It is recommended to put the instruction in the preinitialize event, because binding event won't be triggered after page refreshing or switching.

**Classification II: GUI Designing Command**

https://www.itead.cc/wiki/Nextion_Instruction_Set
Note: When you can’t realize some special GUI designing in Nextion Editor, you can use some GUI commands to make it happen. Generally, the controls in Nextion editor can satisfy your GUI designing demand.

**cls: clear the screen**

cls color

**color**: Decimal color value or color code

- Example 1:

```
cls 1024 //Refresh the screen with decimal 1024 color value
```

- Example 2:

```
cls RED //Refresh the screen with the color of code RED (RED represents red color)
```

Both Decimal color value and Color code are supported in Nextion Editor.

**pic: display picture**

```pic x, y, picid
x: x coordinate starting point;
y: y coordinate starting point;
```

**picid**: Picture ID;

- Example 1:

```
pic 10, 20, 0 //Display the picture (ID is 0) in resource file at the coordinate (10, 20)
```

- Example 2:

```
pic 40, 50, 1 //Display the picture (ID is 1) in resource file at the coordinate (40, 50)
```

**picq: Crop picture**

```picq x, y, w, h, picid
x: x coordinate starting point;
y: y coordinate starting point;
w: area width;
```

https://www.itead.cc/wiki/Nextion_Instruction_Set
h: area height;

picid: Picture ID;

   • Example:

   ```
   pic q 20, 50, 30, 20, 0
   //Crop the area from starting coordinate (20, 50) , with a width 30×height 20 size, 
   //in the picture 0 (the background picture must be full-screen) and display it on
   //the screen, and the display coordinate is the starting coordinate (20, 50).
   ```

Remarks:

This instruction requires that the background picture must be full-screen; otherwise, the image you crop is not the one you want. The crop image area and the display area is overlap on the screen.

---

**xpic: Advanced crop picture**

xpic x, y, w, h, x0, y0, picid

x: x coordinate starting point;

y: y coordinate starting point;

w: Region width;

h: Region height;

x0: crop picture starting point x coordinate;

y0: crop picture starting point y coordinate;

picid: Picture ID;

   • Example:

   ```
   pic q 20, 50, 30, 20, 15, 15, 0
   //Crop the area from starting coordinate (15, 15) , 
   //with a width 30 × height 20 size, in the picture 0 (the background
   //picture must be full-screen) and display it on the screen, and the display
   //coordinate is the starting coordinate (20, 50).
   ```

---

**xstr: Print string on the device**

xstr x, y, w, h, fontid, fontcolor, backcolor, xcenter, ycenter, sta, string
x: x coordinate starting point;
y: y coordinate starting point;
w: area width;
h: area height;
fontid: Font ID;
fontcolor: Font color;
backcolor: Background color (when set sta as Crop Image or Image, backcolor means image ID);
xcenter: Horizontal alignment (0 is left-aligned, 1 is centered, 2 is right-aligned);
ycenter: Vertical alignment (0 is upper-aligned, 1 is centered, 2 is lower-aligned);
sta: Background fill(0-crop image;1-solid color;2-Image; 3-No backcolor, when set sta as Crop Image or Image, backcolor means image ID);
string: Character content;
  - Example:

    xstr 0, 0, 100, 30, 1, RED, BLACK, 1, 1, 1, "China"

  - Explanation:

    Use font 1, at the starting point coordinate(0,0), write"China" in an area that its width is 100, height is 30, the font color is RED, background color is BLACK, horizontal alignment is center, and vertical alignment is center too.

Remarks:

1. There is automatic word wrapping if characters exceeds the default set w. If there are remaining characters not written out when wrapped to h, they will be neglected.

2. For more information about color value, please refer to cls command.

fill x, y, w, h, color

Fill area

x: x coordinate starting point;
y: y coordinate starting point;
w: area width;
h: area height;
color: fill color;

- Example:

```
fill 0, 0, 100, 30, RED  //Fill color RED in the area of starting coordinate (0, 0) and width 100×height 30
```

Remarks:

For more information about color value, please refer to cls command.

---

**line x, y, x2, y2, color**

x: x coordinate starting point;
y: y coordinate starting point;
x2: x coordinate ending point;
y2: y coordinate ending point;
color: Line color;

- Example:

```
line 0, 0, 100, 100, RED  //Draw a line in color RED between the coordinate (0, 0) and the coordinate (100, 100)
```

Remarks:

For more information about color value, please refer to cls command.

---

**draw x, y, x2, y2, color**

x: x coordinate starting point;
y: y coordinate starting point;
x2: x coordinate ending point;
y2: y coordinate ending point;
color: Line color;

- Example:

```
draw 0, 0, 100, 100, RED  //Draw a rectangle, the top left coordinate is (0, 0) and bottom right corner is (100, 100), rectangle frame color is RED.
```

Remarks:
1. What is drawn by draw is hollow rectangle. Please directly use area fill instruction of fill if the filled rectangle needs filling. 2. For more information about color value, please refer to cls command.

`cir x, y, r, color`

cir x, y, r, color: draw a hollow circle
x: Coordinate x of the center of a circle
y: Coordinate y of the center of a circle
r: Radius
color: Line color;

- Example:

```
cir 100, 100, 30, RED    //Draw a hollow circle whose radius is 30 with the coordinate (100, 100) as the center of a circle, circle frame line is RED
```

Remarks: For more information about color value, please refer to cls command.

---

`cirs x, y, r, color`

cirs x, y, r, color: draw a solid circle
x: circle center x coordinate
y: circle center y coordinate
r: Radius
color: Fill color;

- Example

```
cirs 100, 100, 30, RED   //Draw a solid circle whose radius is 30 with the coordinate (100, 100) as the center of a circle, circle color is RED
```

Remarks: For more information about color value, please refer to cls command.

---

**Nextion HMI: System Variables List**
<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Meaning</th>
<th>Instance/Remarks</th>
</tr>
</thead>
</table>
| 1    | dim  | Current value of backlight brightness | 1.\(\text{dim}=50\) 2.\(\text{dim}+10\) 3.\(\text{dim}-10\)  
When you set \(\text{dim}=80\), it means you have set the brightness as 80, but did not save it as default. Next time you power on the Nextion TFT, the brightness of backlight will keep its default setting. |
| 2    | dims | default backlight brightness when Nextion is powered on | 1.\(\text{dims}=50\) 2.\(\text{dims}+10\) 3.\(\text{dims}-10\)  
When you set \(\text{dims}=80\) to Nextion TFT, it means you have set the brightness as 80, and have saved it as default. Next time you power on the Nextion TFT, the brightness of backlight will be 80 by default. |
| 3    | baud | Current value of baud | 1.\(\text{baud}=2400\) 2.\(\text{baud}=4800\) 3.\(\text{baud}=9600\)  
4.\(\text{baud}=19200\) 5.\(\text{baud}=38400\) 6.\(\text{baud}=57600\) 7.\(\text{baud}=115200\) |
| 4    | bauds | Default value of baud when Nextion is powered on | 1.\(\text{bauds}=9600\) is the default baud of factory settings.  
2. when you set \(\text{bauds}=115200\) to Nextion, it means you have set the baud as 115200, and have saved it as default. Next time you power the Nextion, the value of baud will be 115200 by default. |
| 5    | spax | Horizontal spacing of font display (default 0) | \(\text{spax}=2\) |
| 6    | spay | Vertical spacing of font display (default 0) | \(\text{spay}=2\) |
| 7    | thc  | Brush color at touch drawing | 1.\(\text{thc}=\text{RED}\) 2.\(\text{thc}=\text{I024}\) |
| 8    | thdra | Touch drawing function | \(\text{thdra}=0(\text{C1ose})\) \(\text{thdra}=1(\text{Open})\) |
| 9    | ussp | If no serial data, it will auto activate sleep time (unit: second, minimum 3, maximum 65,535, power-on default 0) | \(\text{usssp}=30(\text{No serial data within 30 seconds, it auto will enter into sleep mode})\)  
\(\text{usssp}=0(\text{invalid})\) |
| 10   | thsp | If no touch operation, it will auto enter into sleep time (unit: second, minimum 3, maximum  
\(\text{thsp}=30(\text{No touch operation within 30 seconds, it will auto enter into sleep mode})\)  
\(\text{thsp}=0(\text{invalid})\) |
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 11 | thup | Touch in sleep mode will auto-awake switch (power-on default 0)  
Remarks: Whether thup be 0 or 1, whenever there is any touch operation in sleep mode, the device will send touch coordinates to the serial port.  
thup=0 (Touch will not auto awake switch during sleep mode)  
thup=1 (Touch will auto awake switch during sleep mode) |
| 12 | sendxy | Close or open real-time sending touch coordinate function  
Remarks:  
1. When this function is open, the device will send touch coordinate through serial port when you touch the screen.  
2. Please refer to the table: Format of Device Return Data to learn more about the format of sending the coordinate.  
sendxy=0 (Close)  
sendxy=1 (Open) |
| 13 | delay | Delay=100 (Pause the device for 100ms)  
Remarks: When delay command is executed, the CPU of the device will not execute any commands, but will continue receiving serial port command and store them to buffer. |
| 14 | sleep | sleep=0 (Exit Sleep)  
sleep=1 (Enter Sleep)  
Remarks:  
When the device wake up from sleep mode, the device will auto refresh the current page, and the backlight brightness will recover to the default brightness value. Two commands available for changing the brightness of backlight, dim and dims. |
| 15 | bkcmd | Data return of successful/failed execute serial command (2 by default)  
bkcmd=0 (No return)  
bkcmd=1 (Only return the successful data)  
bkcmd=2 (Only return the failed data)  
bkcmd=3 (Always return)  
Remarks:  
This setting only affects the serial command’s successful execution or fail execution’s data return. In Nextion Editor’s editing interface, when there is a command execution error, it will return error data; when the command being executed successfully, it will not return execution result data. |
<table>
<thead>
<tr>
<th>Page</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
</table>
| 16   | rand     | random value | dim=rand (assign a random value to backlight brightness)  
n0.val=rand (assign a random value to variable of n0.val)  
Remarks:  
1. Before use rand, you should use randset command to set random value generated range. If you do not set it, the default range will be 0~4294967295. Aftering setting randset, you will get a new random value within the preset range every time you run rand.  
2. The range set by randset keeps valid unless the device being reboot or reset. |
| 17   | sys0     | Numeric system variables | sys0=10  
sys1=40  
sys2=60  
n.val=sys2  
Remarks:  
ys0, sys1, sys2 are global variables, which do not required to define or create. You can use them in any page. Default value for these three variables are 0, they can be read and write, their value range are 0~4294967295. It's recommended to pass values through pages with they system variables. |
| 18   | rtc0 to rtc6 | RTC variables (Enhanced Only) | rtc0, year;  
rtc1, month;  
rtc2, day;  
rtc3, hour;  
rtc4, min;  
rtc5, second;  
rtc6, week; |
| 19   | pio0 to pio7 | GPIO variables (Enhanced Only) | Default mode when power on: pull up input mode  
Internal pull up resistor: 50K |
| 20   | pwm4 to pwm7 | Duty cycle for the PWM GPIO (Enhanced Only) | min value = 0, max value = 100, default = 50; |
| 21   | pwmf     | PWM frequency (Enhanced Only) | min value = 1Hz, max value = 65536 Hz, default = 1000Hz;  
All PWM output is unified to one frequency, no independent setting is allowed. |
<table>
<thead>
<tr>
<th>Code</th>
<th>Decimal System</th>
<th>Indicator Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>RED</td>
<td>63488</td>
<td>Red</td>
</tr>
<tr>
<td>BLUE</td>
<td>31</td>
<td>Blue</td>
</tr>
<tr>
<td>GRAY</td>
<td>33840</td>
<td>Gray</td>
</tr>
<tr>
<td>BLACK</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>WHITE</td>
<td>65535</td>
<td>White</td>
</tr>
<tr>
<td>GREEN</td>
<td>2016</td>
<td>Green</td>
</tr>
<tr>
<td>BROWN</td>
<td>48192</td>
<td>Brown</td>
</tr>
<tr>
<td>YELLOW</td>
<td>65504</td>
<td>Yellow</td>
</tr>
</tbody>
</table>

**Nextion HMI: Color Code List**

**Format of Device Return Data**

**Table 1: serial l instruction execution success or failure notification format**

1. Only when the system variable bkcmd is not zero will return instruction execution succeed or fail data, bkcmd defaults to 0 after each power on, which means it does not return the result of instruction execution.

2. The code of the source file is not affected by bkcmd when software is under editing, the error data will be returned when there is an execution error, and the error data will not be returned when execute success.

3. The returned data is ended with three bytes of “0XFF 0XFF 0XFF”.

wup=255 (default, refresh current page), wup=2 (refresh page 2 when wake up)

Note: Nextion can execute wup even in sleep mode;
<table>
<thead>
<tr>
<th>The first byte of returned data</th>
<th>Meaning</th>
<th>Format</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X00</td>
<td>Invalid instruction</td>
<td>0X00+End</td>
<td>Return this data when receiving the invalid instruction sent by the user</td>
</tr>
<tr>
<td>0X01</td>
<td>Successful execution of instruction</td>
<td>0X01+End</td>
<td>Return this data when the instruction sent by the user is successfully executed</td>
</tr>
<tr>
<td>0X03</td>
<td>Page ID invalid</td>
<td>0X03+End</td>
<td>Return this data when the instruction sent by the user contains invalid page ID or invalid page name</td>
</tr>
<tr>
<td>0X04</td>
<td>Picture ID invalid</td>
<td>0X04+End</td>
<td>Return this data when the instruction sent by the user contains invalid picture ID</td>
</tr>
<tr>
<td>0X05</td>
<td>Font ID invalid</td>
<td>0X05+End</td>
<td>Return this data when the instruction sent by the user contains invalid font ID</td>
</tr>
<tr>
<td>0X11</td>
<td>Baud rate setting invalid</td>
<td>0X11+End</td>
<td>The baud rate setting instruction sent by the user contains invalid baud rate parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Baud rate supported by the device including: 2400 4800 9600 19200 38400 57600 115200</td>
</tr>
<tr>
<td>0X12</td>
<td>Curve control ID number or channel number is invalid</td>
<td>0X12+End</td>
<td>When users use the add commands to add data to curve, this data will be returned as curve control ID number or channel number is invalid</td>
</tr>
<tr>
<td>0X1A</td>
<td>Variable name invalid</td>
<td>0X1A+End</td>
<td>Return this data when the serial port receives invalid variable name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: control attribute is also called variable. For example, when you set the attribute of a control, it will return this data if the unavailable attribute name is input.</td>
</tr>
<tr>
<td>0X1B</td>
<td>Variable operation invalid</td>
<td>0X1B+End</td>
<td>For example, when txt attribute of text component t0 is assigned, it should be written as t0.txt=“abc” It is wrong if you write t0.txt=abc. For another example, the val attribute of progress bar j0 should be numerical, so it should be written as j0.val=50; it will be wrong if you write j0.val=“50” or j0.val=abc.</td>
</tr>
<tr>
<td>0X1C</td>
<td>Failed to assign</td>
<td>0X1C+End</td>
<td>Return this data when attribute failed to assign.</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Result</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td>-----------------</td>
<td></td>
</tr>
<tr>
<td>0X1D</td>
<td>Operate PERFROM failed</td>
<td>0X1D+End</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return this data when opearete PERFROM failed.</td>
<td></td>
</tr>
<tr>
<td>0X1E</td>
<td>Parameter quantity invalid</td>
<td>0X1E+End</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return this data when the instruction parameter quantity input by user is wrong.</td>
<td></td>
</tr>
<tr>
<td>0X1F</td>
<td>IO operate failed</td>
<td>0X1F+End</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Return this data when operate IO failed.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2: other data return format**

1. The returned data is end with three bytes of “0xFF 0xFF 0xFF”.
2. The following data’s return does not be affected by bkcmb.
<table>
<thead>
<tr>
<th>The first byte of returned data</th>
<th>Meaning</th>
<th>Format</th>
</tr>
</thead>
</table>
| 0x65                           | Touch event return data | 0x65+Page ID+Component ID+TouchEvent+End  
Return this data when the touch event created by the user is pressed.  
Definition of TouchEvent: Press Event 0x01, Release Event 0x00  
**Instance: 0x65 0x00 0x02 0x01 0xFF 0xFF 0xFF**  
Meaning: Page 0, Button 2, Press |
| 0x66                           | Current page ID number returns | 0x66+Page ID+End  
The device returns this data after receiving “sendme” instruction)  
**Instance: 0x66 0x02 0xFF 0xFF 0xFF**  
Meaning: Current page ID is 2 |
| 0x67                           | Touch coordinate data returns | 0x67++ Coordinate X High-order+Coordinate X Low-order+Coordinate Y High-order+Coordinate Y Low-order+TouchEvent State+End  
When the system variable “sendxy” is 1, return this data at TouchEvent occurring  
Definition of TouchEvent: Press Event 0x01, Release Event 0x00  
**Instance: 0x67 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF**  
Meaning: Coordinate (122,30), Touch Event: Press |
| 0x68                           | Touch Event in sleep mode | 0x68++Coordinate X High-order+Coordinate X Low-order+Coordinate Y High-order+Coordinate Y Low-order+TouchEvent State+End  
When the device enters sleep mode, return this data at TouchEvent occurring  
Definition of TouchEvent: Press Event 0x01, Release Event 0x00  
**Instance: 0x68 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF**  
Meaning: Coordinate (122,30), Touch Event: Press |
| 0x70                           | String variable data returns | 0x70+Variable Content in ASCII code+End  
When the variable obtained through get command is string type, return this data  
**Instance: 0x70 0x61 0x62 0x63 0xFF 0xFF 0xFF**  
Meaning: Return the string data: “abc” |
| 0x71                           | Numeric variable data returns | 0x71+variable binary data(4 bytes little endian mode, low in front)+End  
When the variable obtained by get command is value, this data returns.  
**Instance:0x71 0x66 0x00 0x00 0x00 0xFF 0xFF 0xFF**  
Meaning:return value data:102 |
| 0x86                           | Device automatically enters into sleep mode | 0x86+End  
Only when the device automatically enters into sleep mode will return this data. If execute serial command “sleep = 1” to enter into sleep mode, it will not return this data. |
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0X87</td>
<td>Device automatically wake up</td>
<td>0X87+End Only when the device automatically wake up will return this data. If execute serial command “sleep=0” to wake up, it will not return this data.</td>
</tr>
<tr>
<td>0X88</td>
<td>System successful start up</td>
<td>This data is sent after a successful power-on initialization on the device</td>
</tr>
<tr>
<td>0X89</td>
<td>Start SD card upgrade</td>
<td>This data is sent after the device power on and detect SD card, and then enter upgrade interface</td>
</tr>
<tr>
<td>OXFE</td>
<td>Data transparent transmit ready</td>
<td>The device will enter into transparent transmission data initialization mode after receiving data transparent transmission instruction. Data will be sent once the initialization is completed, which means it has entered into data transparent transmission mode and start to transparent transmit data.</td>
</tr>
</tbody>
</table>

**Useful link**

**Discuss Nextion here:** [http://support.iteadstudio.com/discussions/1000058038](http://support.iteadstudio.com/discussions/1000058038)


- This page was last modified on 28 October 2016, at 09:28.
- This page has been accessed 144,943 times.