LTS: Nextion Instruction Set

These are the set of commands that Nextion can use.

They are categorized into only a few categories

1 – General Rules and Practices

No.	General Rule or Practice
1	**All instructions over serial: *are terminated with three bytes of* 0xFF 0xFF 0xFF** ie: decimal: 255 or hex: 0xFF or ansichar: ÿ or binary: 11111111 ie byte ndt[3] = {255,255,255}; write(ndt,3); or print("\xFF\xFF\xFF"); or print("ÿÿÿ")
2	All instructions and parameters are in ASCII
3	All instructions are in lowercase letters
4	Blocks of code and enclosed within braces { } can not be sent over serial this means if, for, and while commands can not be used over serial
5	A space char 0x20 is used to separate command from parameters.
6	There are no spaces in parameters unless specifically stated
7	Nextion uses integer math and does not have real or floating support.
8	Assignment are non-complex evaluating fully when reaching value after operator.
9	Comparison evaluation is non-complex. Use nesting when and as required.
10	Instructions over serial are processed on receiving termination (see 1.1)

11 Character escaping is limited to 0x0D by using two text chars \r

12 Nextion does not support order of operations. sys0=3+(8*4) is invalid.

13 16-bit 565 Colors are in decimal from 0 to 65535 (see 5.Note)

14 Text values must be encapsulated with double quotes: ie “Hello”

15 Items within specific to Enhanced Models are noted with ***K***

16 Transparent Data Mode (used by addt and wept commands)

1. MCU sending to Nextion
    1. MCU sends command. ie: wept 30,20ÿÿÿ or addt 1,0,320ÿÿÿ
    2. Nextion requires ~5ms to prepare for transparent mode data transfer
    3. Nextion sends “Ready” 0xFE 0xFF 0xFF 0xFF Return Data (see 7.30)
    4. MCU can now send specified quantity (20) of raw bytes to Nextion
    5. Nextion receives raw bytes from MCU until specified quantity (20) is received
    6. Nextion sends “Finished” 0xFD 0xFF 0xFF 0xFF Return Data (see 7.29)
    7. MCU and Nextion can proceed to next command

Note: Nextion will remain waiting at step 5 until every byte of specified quantity is received.
– During this time Nextion can not execute any other commands, and may indeed hang if the MCU fails to deliver the number of bytes as specified by the command parameter.
– data quantity limited by serial buffer (all commands+terminations + data < 1024)

17 Only component attributes in green and non readonly system variables can be assigned new values at runtime. All others are readonly at runtime with the exception of .objname

18 Numeric values can now be entered with byte-aligned hex. ie: n0.val=0x01FF

19 **Advanced**. Address mode is an advanced technique prepending the serial instruction with two bytes for the address. Two byte address is to be sent in little endian order, ie: 2556 is sent 0xFC 0x09. By default, the Nextion address is 0 and does not require two byte prefixing. When the two byte addressing is used, Nextion will only respond to the command if its address matches the two byte prefix, or the transmitted address is 65535 broadcast. See the addr system variable.

20 **Advanced**. Protocol Reparse mode is an advanced technique that allows users to define their own incoming protocol and incoming serial data handling. When in active Protocol Reparse mode, incoming serial data will not be processed natively by the Nextion firmware but will wait in the serial buffer for processing. To exit active Protocol Reparse mode, recmod must be set back to passive (ie: in Nextion logic as recmod=0), which can not be achieved via serial. Send DRAKJHSUYDGBNCJHGJKSHBDNÿÿÿ via serial to exit active mode serially. Most HMI applications will not require Protocol Reparse mode and should be skipped if not fully understood.

21 Commenting user code inside Events uses the double-slash (two characters of forward slash /) technique. See 2.27 for proper usage.

2 – Assignment Statements

No.	Data Type	Operator	Description/Example (see 1.8 and 1.17)
1	Text	=	Assignment. Right side will be evaluated with result placed in left side. Component .txt-maxl needs to be large enough to hold result. t0.txt=”Hello”
2	Text	+=	Text Addition. Will concatenate left side with right side with result placed left side. ie t0.txt+=”Hello” is equivalent to t0.txt=t0.txt+”Hello”. t0.txt=”-“+t0.txt becomes t0.txt=t0.txt+”-“. Use temp variable to prepend. va0.txt=t0.txt t0.txt=”-“+va0.txt t0.txt+=” World” // append ” World” to t0.txt //When contents of t0.txt is “Hello” becomes “Hello World”
3	Text	-=	Text Subtraction. Will remove right side (a specified numeric amount of characters to remove) from end of left side and result placed in left side. t0.txt-=4 or t0.txt=t0.txt-4 // remove last 4 chars from t0.txt
4	Text	\	Escape Character. Supported is \r hex 0x0D. (see 1.11) t0.txt=”\r”
5	Text	==	Boolean Equality. Evaluate left side to right side. If both left and right sides are the same creates a true condition if(t0.txt==va0.txt)
6	Text	!=	Boolean Inequality. Evaluate left side to right side. If both left and right sides are different creates a true condition if(t0.txt!=va0.txt)
7	Numeric	=	Assignment. Right side of equation will be evaluated and result placed in left side. If more than one operator on right side, full evaluation and assignment will occur at each operator. n0.val=bauds // places bauds value in n0.val component
8	Numeric	+=	Numeric Addition. Adds value of left side and right side with result placed in left side. n0.val+=4 is equivalent to n0.val=n0.val+4 n0.val+=va0.val
9	Numeric	-=	Numeric Subtraction. Subtracts right side from left side with result placed in left side. n0.val-=4 is equivalent to n0.val=n0.val-4 n0.val-=60 // decreases value of n0.val by 60
10	Numeric	*=	Numeric Multiplication. Multiplies left side with right side with product result placed in left side. n0.val*=2 is equivalent of n0.val=n0.val*2 n0.val*=2
11	Numeric	/=	Numeric Division. Equates division of numerator (left side) and divisor (right side) and places integer quotient in left side. 60000/20001=2 n0.val/=60
12	Numeric	%=	Numeric Modulo. Equates division of numerator (left side) and divisor (right side) and places integer remainder in left side. Divisor MUST be a constant. 60000/20001=19998 n0.val%=60

13	Numeric	<<	Arithmetic Bit Shift Left. Moves all bits specified number to the left. Using the 16-bit example that follows, (32-bit uses similar rules) All bits shifted above 15 are **lost** and **undefined** bits become 0 n0.val=n0.val<<4 **0 0 0 0.0 0 1 1.1 1 0 0.0 0 0 1** **0 0 1 1.1 1 0 0.0 0 0 1.** **0 0 1 1.1 1 0 0.0 0 0 1.0 0 0 0**
14	Numeric	>>	Arithmetic Bit Shift Right. Moves all bits specified number to the right. Using the 16-bit example that follows, (32-bit uses similar rules) All bits shifted below 0 are **lost** and **undefined** bits become the signed bit. When signed bit is 1 (value is negative) then left filled is with 1's When signed bit is 0 (value is positive) then left filled is with 0's n0.val=n0.val>>4 **0 0 0 0.0 0 1 1.1 1 0 0.0 0 0 1** **0 0 0 0.0 0 1 1.1 1 0 0** **0 0 0 0.0 0 0 0.0 0 1 1.1 1 0 0**
15	Numeric	&	Logical Bitwise AND. Compares all bits left side to all bits right side(mask) Using the 16-bit example that follows, (32-bit uses similar rules) Result is a bit of 1 where both left and right bits were 1 n0.val=n0.val&35381 **0 1 0 1.1 0 1 1.0 0 1 0.0 1 0 1** n0.val of 23333 **1 0 0 0.1 0 1 0.0 0 1 1.0 1 0 1** mask of 35381 **0 0 0 0.1 0 1 0.0 0 1 0.0 1 0 1** result is 2597
16	Numeric	|	Logical Bitwise OR. Compares all bits left side to all bits right side Using the 16-bit example that follows, (32-bit uses similar rules) Result is a bit of 1 where either left or right bits were 1 n0.val=n0.val|35381 **0 1 0 1.1 0 1 1.0 0 1 0.0 1 0 1** n0.val of 65519 **1 0 0 0.1 0 1 0.0 0 1 1.0 1 0 1** constant 35381 **1 1 0 1.1 0 1 1.0 0 1 1.0 1 0 1** result is 56117
17	Numeric	==	Boolean Equality. Evaluate left side to right side. If both left and right sides are the same creates a true condition if(n0.val==va0.val)
18	Numeric	!=	Boolean Inequality. Evaluate left side to right side. If both left and right sides are different creates a true condition if(n0.val!=va0.val)
19	Numeric	<	Boolean Less than. Evaluate left side to right side. If left side is less than right side creates a true condition while(n0.val<va0.val)

20	Numeric	<=	Boolean Less than or Equal. Evaluate left side to right side. If left side is less than or equal to right side creates a true condition while(n0.val<=va0.val)
21	Numeric	>	Boolean Greater than. Evaluate left side to right side. If left side is greater than right side creates a true condition while(n0.val>va0.val)
22	Numeric	>=	Boolean Greater than or Equal. Evaluate left side to right side. If left side is greater than or equal to right side creates a true condition while(n0.val>=va0.val)
23	Code	{ }	Code Block begins with open brace { on line by itself Code Block ends with closing brace } beginning a new line see if (see 3.25) while (see 3.26) and for (see 3.27)
24	Code	( )	Condition enclosure begins with open parenthesis (and ends with closing parenthesis) at end of line see if (see 3.25) while (see 3.26) and for (see 3.27)
25	Code	.	Period Separator. Separates Page, Component and Attributes Also used with page index and component array. (see 2.26) page1.va0.val or p0.pic
26	Code	[ ]	Array[index]. There are 3 arrays. Keyboard source showcases 2 arrays. The b[.id] component array which takes component .id as index The p[index] page array which takes page index as index These (p[].b[]) need to be used with caution and mindful purpose. Reference to a component without specified Attribute can create for long and potentially frustrating debug sessions. The third array is the Serial Buffer Data u[index] array. This is valid when in active Protocol Reparse mode. Protocol Reparse is an advanced technique that should be skipped if not fully understood. p[pageindex].b[component.id].attribute // global scope b[component.id].attribute // local scope on current page
27	Comment	//	Double-Slash Commenting to add user comments to code. Everything to the right of, and including, the double-slash is a comment that will not be executed by the Nextion interpreter. Comments should: 1) occur on a line by themselves with the double-slash at the beginning of the line (no leading spaces), 2) immediately following code on a line without a space separating code and the double slash. Commenting of code blocks should occur: 1) before the condition/iteration 2) inside the opening and closing braces 3) after the code block. *Notes:* It is important to note that comments can not interrupt code blocks without causing an “Error: Index was outside the bounds of the array”. Comments are counted towards the overall “code + attributes” hard limit of 65534.

```
// these are valid comments
sys0=0// reset sys0 to zero
```

The following showcases valid/invalid use

```
//valid comment before condition/iteration
for(sys0=0;sys0<=sys1;sys0++)
// invalid comment between condition/iteration and block
{
  doevents//valid comment after code on same line
  // valid comment inside block
}
// valid comment outside block
```

3 – Operational Commands

No.	Name	Param Count	Description and Usage/Parameters/Examples
1	page	1	Change page to page specified. Unloads old page to load specified page. Nextion loads page 0 by default on power on. usage: page <pid> <pid> is either the page index number, or pagename page 0 // Change page to indexed page 0 page main // Change page to the page named main
2	ref	1	Refresh component (auto-refresh when attribute changes since v0.38) – if component is obstructed (stacking), ref brings component to top. usage: ref <cid> <cid> is component's .id or .objname attribute of component to refresh – when <cid> is 0 (page component) refreshes all on the current page. ref t0 // Refreshes the component with .objname of t0 ref 3 // Refreshes the component with .id of 3 ref 0 // Refreshes all components on the current page (same as ref 255)
3	click	2	Trigger the specified components Touch Press/Release Event As event code is always local, object can not be page prefixed usage: click <cid>,<event> <cid> is component's .id or .objname attribute of component to refresh <event> is 1 to trigger Press Event, 0 to trigger Release Events click b0,1 // Trigger Touch Press Event of component with .objname b0 click 4,0 // Trigger Touch Release Event of component with .id 4
4	ref_stop	0	Stops default waveform refreshing (will not refresh when data point added) – waveform refreshing will resume with ref_star (see 3.5) usage: ref_stop ref_stop // stop refreshing the waveform on each data point added

5	ref_star	0	Resume default waveform refreshing (refresh on data point add) – used to resume waveform refreshing stopped by ref_stop (see 3.4) usage: ref_start ref_star // resume default refreshing, refresh on each data point added
6	get	1	Send attribute/constant over serial (0x70/0x71 Return Data) usage: get <attribute> <attribute> is either numeric value, .txt contents, or constant get t0.txt // sends text contents of t0.txt in 0x70 Return Data format get “123” // sends text constant “123” in 0x70 Return Data format get n0.val // sends numeric value of n0.val in 0x71 Return Data format get 123 // sends numeric constant 123 in 0x71 Return Data format
7	sendme	0	Sends number of currently loaded page over serial (0x66 Return Data) – number of currently loaded page is stored in system variable dp – used in a page’s initialize event will auto-send as page loads usage: sendme sendme // sends the value of dp in 0x66 Return Data Format
8	cov	3	Convert variable from numeric type to text, or text to numeric type – text must be text ASCII representation of an integer value. – source and destination types must not be of the same type – when length is fixed and value is less, leading zeros will be added ie: src numeric value of 123 with length 4, result is dest text “0123” – dest txt_maxl and length needs be large enough to accommodate conversion. ie: src numeric value of 123 with length 0, result is dest text “123” – when value is larger than length, .txt **results in a truncation** – it is recommended to handle source length in user code before cov **Note:** v0.53 changed behaviour from previous pre v0.53 behaviours. The pre v0.53 behaviour has been restored in the LTS version. cov is deemed undefined if source is larger than length or the dest txt_maxl is smaller than the requested length. Some undefines are exploitable. usage: cov <src>,<dest>,<length> <src> is text attribute (or numeric attribute when <dest> is text) <dest> is numeric attribute (or text attribute when <src> is numeric) <length> will determine if leading zeros added to conversion to text cov h0.val,t0.txt,0 // convert value of h0 into t0.txt without leading zeros cov t0.txt,h0.val,0 // convert integer into t0.txt from h0.val <length> ignored. cov h0.val,t0.txt,4 // convert value of h0 into t0.txt with exactly 4 digits Invalid: cov h0.val,va0.val,0 or cov t0.txt,va0.txt,0 // src & dest same type.
8a	covx	4	Convert variable from numeric type to text, or text to numeric type – text must be text ASCII representation of an integer value. – source and destination types must not be of the same type

– when source is numeric, hex format and length not 0 and <4.
ie: (len 2) positive significant (byte 0 to 3), 123 = 0000007B = 007B
ie: (len 2) negative significant (byte 3 to 0), -123 = FFFFFF85 = FF85
– value is more than allowed space **results in a truncation**
– it is recommended to ensure handling source length in user code before covx
– in v0.53, covx is deemed undefined if source is larger than length or
dest txt_maxl is smaller than requested length.
(some of these undefines, can be exploited)
ie: src numeric value of 123 with length 0, result is dest text “123”
– when length is fixed and value is less, leading zeros will be added
ie: src numeric value of 123 with length 4, result is dest text “0123”
– when value is larger than length, .txt truncated to least significant digits
ie: src numeric value of 23425 with length 4 result is dest text “3425”
usage: covx <src>,<dest>,<length>,<format>
<src> is text attribute (or numeric attribute when <dest> is text)
<dest> is numeric attribute (or text attribute when <src> is numeric)
<length> will determine if leading zeros added to conversion to text
<format> 0: integer, 1: Comma separated 1,000s, 2: Hex
covx h0.val,t0.txt,0,0 // convert value of h0 into t0.txt without leading zeros
covx t0.txt,h0.val,0,0 // convert t0.txt into integer in h0.val <length> ignored.
covx h0.val,t0.txt,4,0 // convert value of h0 into t0.txt with exactly 4 digits
covx h0.val,t0.txt,4,1 // convert value of h0 into t0.txt with commas
covx h0.val,t0.txt,4,2 // convert value of h0 into t0.txt in 2 bytes of hex digits
Invalid: cov h0.val,va0.val,0 or cov t0.txt,va0.txt,0 // src & dest same type.

9 touch_j 0 Recalibrate the Nextion device’s touch sensor
– presents 4 points on screen for user to touch, saves, and then reboots.
– typically not required as device is calibrated at factory
– sensor can be effected by changes of conditions in environment
usage: touch_j
touch_j // trigger the recalibration of touch sensor

10 substr 4 Extract character or characters from contents of a Text attribute
usage: substr <src>,<dest>,<start>,<count>
<src> is text attribute where text will be extracted from
<dest> is text attribute to where extracted text will be placed
<start> is start position for extraction (0 is first char, 1 second)
<count> is the number of characters to extract
substr va0.txt,t0.txt,0,5 // extract first 5 chars from va0.txt, put into t0.txt

11 vis 2 Hide or Show component on current page
– show will refresh component and bring it to the forefront layer
– hide will remove component visually, touch events will be disabled

			– use layering with mindful purpose, can cause ripping and flickering. – use with caution and mindful purpose, may lead to difficult debug session usage: vis <comp><state> <comp> is either .objname or .id of component on current page, – valid .id is 0 – page, 1 to 250 if component exists, and 255 for all <state> is either 0 to hide, or 1 to show. vis b0,0 // hide component with .objname b0 vis b0,1 // show component with .objname b0, refresh on front layer vis 1,0 // hide component with .id 1 vis 1,1 // show component with .id 1, refresh on front layer vis 255,0 // hides all components on the current page
12	tsw	2	Enable or disable touch events for component on current page – by default, all components are enabled unless disabled by tsw – use with caution and mindful purpose, may lead to difficult debug session usage: tsw <comp><state> <comp> is either .objname or .id of component on current page, – valid .id is 0 – page, 1 to 250 if component exists, and 255 for all <state> is either 0 to disable, or 1 to enable. tsw b0,0 // disable Touch Press/Release events for component b0 tsw b0,1 // enable Touch Press/Release events for component b0 tsw 1,0 // disable Touch Press/Release events for component with id 1 tsw 1,1 // enable Touch Press/Release events for component with id 1 tsw 255,0 // disable all Touch Press/Release events on current page
13	com_stop	0	Stop execution of instructions received from Serial – Serial will continue to receive and store in buffer. – execution of instructions from Serial will resume with com_star (see 3.14) – using com_stop and com_star may cause a buffer overflow condition. – Refer to device datasheet for buffer size and command queue size usage: com_stop com_stop // stops execution of instructions from Serial
14	com_star	0	Resume execution of instructions received from Serial – used to resume an execution stop caused by com_stop (see 3.13) – when com_star received, all instructions in buffer will be executed – using com_stop and com_star may cause a buffer overflow condition. – Refer to device datasheet for buffer size and command queue size usage: com_star com_star // resume execution of instruction from Serial
15	randset	2	Set the Random Generator Range for use with rand (see 6.14) – range will persist until changed or Nextion rebooted – set range to desired range before using rand

			– power on default range is -2147483648 to 2147483647, runtime range is user definable. usage: randset <min>,<max> <min> is value is -2147483648 to 2147483647 <max> is value greater than min and less than 2147483647 randset 1,100 //set current random generator range from 1 to 100 randset 0,65535 //set current random generator range from 0 to 65535
16	code_c	0	Clear the commands/data queued in command buffer without execution usage: code_c code_c // Clears the command buffer without execution
17	print	1	Send raw formatted data over Serial to MCU – print/printh does not use Nextion Return Data, user must handle MCU side – qty of data may be limited by serial buffer (all data < 1024) – numeric value sent in 4 byte 32-bit little endian order value = byte1+byte2*256+byte3*65536+byte4*16777216 – text content sent is sent 1 ASCII byte per character, without null byte. usage: print <attr> <attr> is either component attribute, variable or Constant print t0.txt // return 1 byte per char of t0.txt without null byte ending. print j0.val // return 4 bytes for j0.val in little endian order print “123” // return 3 bytes for text “123” 0x31 0x32 0x33 print 123 // return 4 bytes for value 123 0x7B 0x00 0x00 0x00
17a	prints	2	Send raw formatted data over Serial to MCU – prints does not use Nextion Return Data, user must handle MCU side – qty of data may be limited by serial buffer (all data < 1024) – numeric value sent in 4 byte 32-bit little endian order value = byte1+byte2*256+byte3*65536+byte4*16777216 – text content sent is sent 1 ASCII byte per character, without null byte. usage: prints <attr>,<length> <attr> is either component attribute, variable or Constant <length> is either 0 (all) or number to limit the bytes to send. prints t0.txt,0 // return 1 byte per char of t0.txt without null byte ending. prints t0.txt,4 // returns first 4 bytes, 1 byte per char of t0.txt without null byte ending. prints j0.val,0 // return 4 bytes for j0.val in little endian order prints j0.val,1 // returns 1 byte of j0.val in little endian order prints “123”,2 // return 2 bytes for text “12” 0x31 0x32 prints 123,2 // returns 2 bytes for value 123 0x7B 0x00
18	printh	1 to many	Send raw byte or multiple raw bytes over Serial to MCU – printh is one of the few commands that parameter uses space char 0x20 – when more than one byte is being sent a space separates each byte – byte is represented by 2 of (ASCII char of hexadecimal value per nibble)

– qty may be limited by serial buffer (all data < 1024)

– print/printh does not use Nextion Return Data, user must handle MCU side

usage: printh <hexhex>[<space><hexhex][…<space><hexhex]

<hexhex> is hexadecimal value of each nibble. 0x34 as 34

<space> is a space char 0x20, used to separate each <hexhex> pair

printh 0d // send single byte: value 13 hex: 0x0d

printh 0d 0a // send two bytes: value 13,10 hex: 0x0d0x0a

19 add 3 Add single value to Waveform Channel

– waveform channel data range is min 0, max 255

– 1 pixel column is used per data value added

– y placement is if value < s0.h then s0.y+s0.h-value, otherwise s0.y

usage: add <waveform>,<channel>,<value>

<waveform> is the .id of the waveform component

<channel> is the channel the data will be added to

<value> is ASCII text of data value, or numeric value

– valid: va0.val or sys0 or j0.val or 10

add 1,0,30 // add value 30 to Channel 0 of Waveform with .id 1

add 2,1,h0.val // add h0.val to Channel 1 of Waveform with .id 2

20 addt 3 Add specified number of bytes to Waveform Channel over Serial from MCU

– waveform channel data range is min 0, max 255

– 1 pixel column is used per data value added.

– addt uses Transparent Data Mode (see 1.16)

– waveform will not refresh until Transparent Data Mode completes.

– qty limited by serial buffer (all commands+terminations + data < 1024)

– also refer to add command (see 3.19)

usage: add <waveform>,<channel>,<qty>

<waveform> is the .id of the waveform component

<channel> is the channel the data will be added to

<qty> is the number of byte values to add to <channel>

addt 2,0,20 // adds 20 bytes to Channel 0 Waveform with .id 2 from serial

// byte of data is not ASCII text of byte value, but raw byte of data.

21 cle 3 Clear waveform channel data

usage: cle <waveform>,<channel>

<waveform> is the .id of the waveform component

<channel> is the channel to clear

<channel> must be a valid channel: < waveform.ch or 255

0 if .ch 1, 1 if .ch 2, 2 if .ch 3, 3 if .ch=4 and 255 (all channels)

cle 1,0 // clear channel 0 data from waveform with .id 1

cle 2,255 // clear all channels from waveform with .id 2

22	rest	0	Resets the Nextion Device usage: rest rest // immediate reset of Nextion device – reboot.
23	doevents	0	Force immediate screen refresh and receive serial bytes to buffer – useful inside exclusive code block for visual refresh (see 3.26 and 3.27) usage: doevents doevents // allows refresh and serial to receive during code block
24	strlen	2	Computes the length of string in <txt> and puts value in <len> usage: strlen <txt>,<len> <txt> must be a string attribute ie: t0.txt, va0.txt <len> must be numeric ie: n0.val, sys0, va0.val strlen t0.txt,n0.val // assigns n0.val with length of t0.txt content
24a	btlen	2	Computes number of bytes string in <txt> uses and puts value in <len> usage: btlen <txt>,<len> <txt> must be a string attribute ie: t0.txt, va0.txt <len> must be numeric ie: n0.val, sys0, va0.val btlen t0.txt,n0.val // assigns n0.val with number of bytes t0.txt occupies
25	if	Block	Conditionally execute code block if boolean condition is met – execute commands within block { } if (condition) is met. – nested conditions () is not allowed. invalid: ((h0.val+3)>0) – block opening brace { must be on line by itself – no space between block close brace } and else. valid: }else invalid: } else – Text comparison supports ==, != – Numerical comparison supports >, <, ==, !=, >=, <= – nested “if” and “else if” supported. usage: if condition block [else if condition block] [else block] – (condition) is a simple non-complex boolean comparison evaluation valid: (j0.val>75) invalid: (j0.val+1>75) invalid: (j0.val<now.val+60)

```
if(t0.txt=="123456")
{
  page 1
}

if(n0.val==123)
{
  b0.txt="stop"
}else
{
  b0.txt="start"
}

if(rtc==1)
{
  t0.txt="Jan"
}else if(rtc1==2)
{
  t0.txt="Feb"
}else if(rtc1==3)
{
  t0.txt="Mar"
}else
{
  t0.txt="etc"
}
```

26 while Block Continually executes code block until boolean condition is no longer met

– tests boolean condition and execute commands within block { } if condition was met and continues to re-execute block until condition is not met.

– nested conditions () is not allowed. invalid: ((h0.val+3)>0)

– block opening brace { must be on line by itself

– Text comparison supports ==, !=

– Numerical comparison supports >, <, ==, !=, >=, <=

– block runs exclusively until completion unless doevents used (see 3.23)

usage: while condition block

– (condition) is a simple non-complex boolean comparison evaluation

valid: (j0.val>75) invalid: (j0.val+1>75)

```
// increment n0.val as lon as n0.val < 100.  result: b0.val=100
// will not visually see n0.val increment, refresh when while-loop completes
while(n0.val<100)
{
  n0.val++
}

//increment n0.val as long as n0.val < 100. result: n0.val=100
// will visually see n0.val increment, refresh each evaluation of while-loop
while(n0.val<100)
{
  n0.val++
  doevents
}
```

27 for Block Iterate execution of code block as long as boolean condition is met

– executes init_assignment, then tests boolean condition and executes commands within block { } if boolean condition is met, when iteration of block execution completes step_assignment is executed. Continues to iterate block and step_assignment until boolean condition is not met.

– nested conditions () is not allowed. invalid: ((h0.val+3)>0)

– block opening brace { must be on line by itself

– Text comparison supports ==, !=

– Numerical comparison supports >, <, ==, !=, >=, <=

– block runs exclusively until completion unless doevents used (see 3.23)

usage: for(init_assignment;condition;step_assignment) block

– init_assignment and step_assignment are simple non-complex statement

valid: n0.val=4, sys2++, n0.val=sys2*4+3 invalid: n0.val=3+(sys2*4)-1

– (condition) is a simple non-complex boolean comparison evaluation

valid: (j0.val>75) invalid: (j0.val+1>75)

```
// iterate n0.val by 1's as long as n0.val<100. result: n0.val=100
// will not visually see n0val increment until for-loop completes
for(n0.val=0;n0.val<100;n0.val++)
{
}

////iterate n0.val by 2's as long as n0.val<100. result: n0.val=100
// will visually see n0.val increment when doevents processed
for(n0.val=0;n0.val<100;n0.val+=2)
{
  doevents
}
```

28	wepo	2	Store value/string to EEPROM
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			– numeric value length: is 4 bytes, -2147483648 to 2147483647
			– numeric data type signed long integer, stored in little endian order.
			val[addr+3]*16777216+val[addr+2]*65536+val[addr+1]*256+val[addr]
			– string content length: .txt content is .txt-maxl +1, or constant length +1
			usage: wepo <attr>,<addr>
			<attr> is variable or constant
			<addr> is storage starting address in EEPROM
			wepo t0.txt,10 // writes t0.txt contents in addresses 10 to 10+t0.txt-maxl
			wepo “abcd”,10 // write constant “abcd” in addresses 10 to 14
			wepo 11,10 // write constant 11 in addresses 10 to 13
			wepo n0.val,10 // write value n0.val in addresses 10 to 13
29	repo	2	Read value from EEPROM
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			– numeric value length: is 4 bytes, -2147483648 to 2147483647
			– numeric data type signed long integer, stored in little endian order.
			val[addr+3]*16777216+val[addr+2]*65536+val[addr+1]*256+val[addr]
			– string content length: .txt content is lesser of .txt-maxl or until null reached.
			usage: repo <attr>,<addr>
			<attr> is variable or constant
			<addr> is storage starting address in EEPROM
			repo t0.txt,10 // reads qty .txt-maxl chars (or until null) from 10 into t0.txt
			repo n0.val,10 // reads 4 bytes from address 10 to 13 into n0.val
30	wept	2	Store specified number of bytes to EEPROM over Serial from MCU
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			– wept uses Transparent Data Mode (see 1.16)
			– qty limited by serial buffer (all commands+terminations + data < 1024)
			usage: wept <addr>,<qty>
			<addr> is storage starting address in EEPROM
			<qty> is the number of bytes to store
			wept 30,20 // writes 20 bytes into EEPROM addresses 30 to 49 from serial
			// byte of data is not ASCII text of byte value, but raw byte of data.

31	rept	2	Read specified number of bytes from EEPROM over Serial to MCU
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			usage: rept <addr>,<qty>
			<addr> is storage starting address in EEPROM
			<qty> is the number of bytes to read
			rept 30,20 // sends 20 bytes from EEPROM addresses 30 to 49 to serial
			// byte of data is not ASCII text of byte value, but raw byte of data.
32	cfgpio	3	Configure Nextion GPIO
K			usage: cfgpio <io><mode><comp>
			<io> is the number of the extended I/O pin.
			– Valid values in PWM output mode: 4 to 7, all other modes 0 to 7.
			<mode> is the working mode of pin selected by <io>.
			– Valid values: 0-pull up input, 1-input binding, 2-push pull output,
			3-PWM output, 4-open drain output.
			<comp> component .objname or .id when <mode> is 1 (otherwise use 0)
			– in binding mode, cfgpio needs to be declared after every refresh of page to reconnect to
			Touch event, best to put cfgpio in page pre-initialization event
			cfgpio 0,0,0 // configures io0 as a pull-up input. Read as n0.val=pio0.
			cfgpio 1,2,0 // configures io1 as a push-pull output, write as pio1=1
			cfgpio 2,1, b0 // configures io2 as binding input with current page b0.
			// binding triggers b0 Press on falling edge and b0 Release on rising edge
			For PWM mode, set duty cycle before cfgpio: ie: pwm4=20 for 20% duty.
			cfgpio 4,3,0 // configures io4 as PWM output. pwmf=933 to change Hz.
			// changing pwmf changes frequency of all configured PWM io4 to io7
33	ucopy	4	**Advanced**. Read Only. Valid in active Protocol Reparse mode.
			Copies data from the serial buffer.
			When Nextion is in active Protocol Reparse mode, ucopy copies data from the serial buffer.
			Most HMI applications will not require Protocol Reparse and should be skipped if not fully
			understood.
			usage: ucopy <attr>,<srcstart>,<len>,<deststart>
			<attr> must be a writeable attribute ie: t0.txt, va0.val
			<srcstart> must be numeric value ie: 0
			<len> must be a numeric value ie: 4
			<deststart> must be numeric value ie: 0
			ucopy n0.val,0,2,0 // copy buffer bytes 0,1 to lower 2 bytes of n0.val
			ucopy n0.val,0,2,2 // copy buffer bytes 0,1 to upper 2 bytes of n0.val
			ucopy n0.val,0,4,0 // copy buffer bytes 0,1,2,4 to n0.val
			ucopy t0.txt,0,10,0 // copy buffer bytes 0 to 9 into t0.txt

4 – GUI Designing Commands

No.	Name	Param Count	Description and Usage/Parameters/Examples
1	cls	1	Clear the screen and fill the entire screen with specified color usage: cls <color> <color> is either decimal 565 Color Value or Color Constant cls BLUE // Clear the screen and fill with color BLUE cls 1024 // Clear the screen and fill with color 1024
2	pic	3	Display a Resource Picture at specified coordinate usage: pic <x>,<y>,<picid> <x> is the x coordinate of upper left corner where picture should be drawn <y> is the y coordinate of upper left corner where picture should be drawn <picid> is the number of the Resource Picture in the HMI design pic 10,20,0 // Display Resource Picture #0 with upper left corner at (10,20) pic 40,50,1 // Display Resource Picture #1 with upper left corner at (40,50)
3	picq	5	Crop Picture area from Resource Picture using defined area – replaces defined area with content from the same area of Resource Picture – Resource Picture should be full screen-size or area might be undefined usage: picq <x>,<y>,<w>,<h>,<picid> <x> is the x coordinate of upper left corner of defined crop area <y> is the y coordinate of upper left corner of defined crop area <w> is the width of the defined crop area <h> is the height of the defined crop area <picid> is the number of the Resource Picture in the HMI design picq 20,50,30,20,0 // crops area 30×20, from (20,50) to (49,69), from Resource Picture 0
4	xpic	7	Advanced Crop Picture crop area from source Resource Picture render at destination coordinate usage: xpic <destx>,<desty>,<w>,<h>,<srcx>,<srcy>,<picid> <destx> is the x coordinate of destination upper left corner <desty> is the y coordinate of destination upper left corner <w> is the width of the defined crop area <h> is the height of the defined crop area <srcx> is the x coordinate of upper left corner of defined crop area <srcy> is the y coordinate of upper left corner of defined crop area <picid> is the number of the Resource Picture in the HMI design xpic 20,50,30,20,15,15,0 // crops area 30×20, from (15,15) to (44,34), // from Resource Picture 0 and renders it with upper left corner at (20,50)

5	xstr	11	Prints text on the Nextion device using defined area for text rendering
			usage: xstr <x>,<y>,<w>,<h>,<font>,<pco>,<bco>,<xcen>,<ycen>,<sta>,
			<text>
			<x> is the x coordinate of upper left corner of defined text area
			<y> is the y coordinate of upper left corner of defined text area
			<w> is the width of the defined text area
			<h> is the height of the defined text area
			<font> is the number of the Resource Font
			<pco> is the foreground color of text (Color Constant or 565 color value)
			<bco> is a) background color of text, or b) picid if <sta> is set to 0 or 2
			<xcen> is the Horizontal Alignment (0 – left, 1 – centered, 2 – right)
			<ycen> is the Vertical Alignment (0 – top/upper, 1 – center, 3 – bottom/lower)
			<sta> is background Fill (0 – crop image, 1 – solid color, 2 – image, 3 – none)
			<text> is the string content (constant or .txt attribute), ie “China”, or va0.txt
			xstr 10,10,100,30,1,WHITE,GREEN,1,1,1,va0.txt
			// use are 100×30 from (10,10) to (109,39) to print contents of va0.txt using
			// Font Resource 1 rendering Green letters on White background with both
			// horizontal and vertical centering and sta set as solid-color.

6	fill	5	Fill a defined area with specified color usage: fill <x>,<y>,<w>,<h>,<color> <x> is the x coordinate of upper left corner of defined fill area <y> is the y coordinate of upper left corner of defined fill area <w> is the width of the defined fill area <h> is the height of the defined fill area <color> is fill color, either decimal 565 Color Value or Color Constant fill 20,20,150,50,1024 // fills area 150×50 from (20,20) to (169,69) with 565 Color 1024.
7	line	5	Draw a line from point to point with specified color usage: line <x1>,<y1>,<x2>,<y2>,<color> <x1> is the x coordinate of the starting point of the line to be drawn <y1> is the y coordinate of the starting point of the line to be drawn <x2> is the x coordinate of the ending point of the line to be drawn <y2> is the y coordinate of the ending point of the line to be drawn <color> is line color, either decimal 565 Color Value or Color Constant line 20,30,170,200,BLUE // draws line in BLUE from (20,30) to (170,200)
8	draw	5	Draw a hollow rectangle around specified area with specified color usage: draw <x1>,<y1>,<x2>,<y2>,<color> <x1> is the x coordinate of the upper left corner of rectangle area <y1> is the y coordinate of the upper left corner of rectangle area

			<x2> is the x coordinate of the lower right corner of rectangle area <y2> is the y coordinate of the lower right corner of rectangle area <color> is line color, either decimal 565 Color Value or Color Constant draw 10,10,70,70,GREEN // draw a Green rectangle around (10,10) to (79,79) // effectively four lines from (x1,y1) to (x2,y1) to (x2,y2) to (x1,y2) to (x1,y1)
9	cir	4	Draw a hollow circle with specified radius and specified color usage: cir <x>,<y>,<radius>,<color> <x> is the x coordinate of the center point for the circle <y> is the y coordinate of the center point for the circle <radius> is the radius in pixels <color> is line color, either decimal 565 Color Value or Color Constant cir 100,100,30,RED // renders a hollow Red circle with circle center at (100,100), // a 30 pixel radius, a 61 pixel diameter, within boundary (70,70) to (130,130).
10	cirs	4	Draw a filled circle with specified radius and specified color usage: cirs <x>,<y>,<radius>,<color> <x> is the x coordinate of the center point for the circle <y> is the y coordinate of the center point for the circle <radius> is the radius in pixels <color> is fill color, either decimal 565 Color Value or Color Constant cir 100,100,30,RED // renders a filled Red circle with center at (100,100), // a 30 pixel radius, a 61 pixel diameter, within boundary (70,70) to (130,130).

5 – Color Code Constants

No.	Constant	565 Color Value	Indicator Color
1	BLACK	0	Black
2	BLUE	31	Blue
3	BROWN	48192	Brown
4	GREEN	2016	Green
5	YELLOW	65504	Yellow
6	RED	63488	Red
7	GRAY	33840	Gray
8	WHITE	65535	White

Note: 16-bit 565 Colors are in decimal values from 0 to 65535

24-bit RGB **11011000 11011000 11011000**

16-bit 565 **11011 110110 11011**

6 – System Variables

No.	Name	Meaning	Example/Description
1	dp	Current Page ID	dp=1, n0.val=dp read: Contains the current page displayed as per the HMI design write: change page to value specified (same effect as page command) min 0, max # of highest existing page in the user's HMI design.
2	dim dims	Nextion Backlight	dim=32, dims=100 Sets the backlight level in percent min 0, max 100, **default** 100 or user defined Note: dim=32 will set the current backlight level to 32%. using dims=32 will set the current backlight level to 32% and save this to be new power on default backlight level, persisting until changed.
3	baud bauds	Nextion Baud Rate	baud=9600, bauds=9600 Sets the Nextion Baud rate in bits-per-second min 2400, max 115200, **default** 9600 or user defined Valid values are: 2400, 4800, 9600, 19200, 38400, 57600, and 115200. Note: baud=38400 will set the current baud rate to 38400 using bauds=38400 will set the current baud rate to 38400 and save this to be new power on default baud rate, persisting until changed. Note: on rare occasions bauds has become lost. It is recommended to specify bauds=9600 in the first page's Preinitialization Event of HMI.
4	spax spay	Font Spacing	spax=2, spay=2 Globally sets the default rendering space: horizontally between font characters with spax additional pixels and vertically between rows (if multi-lined) with spay additional pixels. min 0, max 65535, **default** 0 Note: Components now have their own individual .spax/.spay attributes that are now used to determine spacing for the individual component.
5	thc	Touch Draw Brush Color	thc=RED, thc=1024 Sets the Touch Drawing brush color min 0, max 65535, **default** 0 Valid choices are either color constants or the decimal 565 color value.
6	thdra	Touch Drawing	thdra=1 (on), thdra=0 (off) Turns the internal drawing function on or off. min 0, max 1, **default** 0 When the drawing function is on, Nextion will follow touch dragging with the current brush color (as determined by the thc variable).
7	ussp	Sleep on No Serial	ussp=30 Sets internal No-serial-then-sleep timer to specified value in seconds min 3, max 65535, **default** 0 (max: 18 hours 12 minutes 15 seconds) Nextion will auto-enter sleep mode if and when this timer expires. Note: ussp=0 is an invalid value, meaning once ussp is set, it will persist and can not be unset unless through reboot or reset.

8	thsp	Sleep on No Touch	thsp=30 Sets internal No-touch-then-sleep timer to specified value in seconds min 3, max 65535, **default** 0 (max: 18 hours 12 minutes 15 seconds) Nextion will auto-enter sleep mode if and when this timer expires. Note: thsp=0 is an invalid value, meaning once thsp is set, it will persist and can not be unset unless through reboot or reset.
9	thup	Auto Wake on Touch	thup=0 (do not wake), thup=1 (wake on touch) Sets if Nextion should auto-wake from sleep when touch press occurs. min 0, max 1, **default** 0 When value is 1 and Nextion is in sleep mode, the first touch will only trigger the auto wake mode and not trigger a Touch Event. thup has no influence on sendxy, sendxy will operate independently.
10	sendxy	RealTime Touch Coordinates	sendxy=1 (start sending) sendxy=0 (stop sending) Sets if Nextion should send 0x67 and 0x68 Return Data min 0, max 1, **default** 0 – Less accurate closer to edges, and more accurate closer to center. Note: expecting exact pixel (0,0) or (799,479) is simply not achievable.
11	delay	Delay	delay=100 Creates a halt in Nextion code execution for specified time in ms min 0, max 65535 As delay is interpreted, a total halt is avoided. Incoming serial data is received and stored in buffer but not be processed until delay ends. If delay of more than 65.535 seconds is required, use of multiple delay statements required. delay=-1 is max. 65.535 seconds.
12	sleep	Sleep	sleep=1 (Enter sleep mode) or sleep=0 (Exit sleep mode) Sets Nextion mode between sleep and awake. min 0, max 1, or **default** 0 When exiting sleep mode, the Nextion device will auto refresh the page (as determined by the value in the wup variable) and reset the backlight brightness (as determined by the value in the dim variable).
13	bkcmd	Pass / Fail Return Data	bkcmd=3 Sets the level of Return Data on commands processed over Serial. min 0, max 3, **default** 2 – Level 0 is Off – no pass/fail will be returned – Level 1 is OnSuccess, only when last serial command successful. – Level 2 is OnFailure, only when last serial command failed – Level 3 is Always, returns 0x00 to 0x23 result of serial command. Result is only sent after serial command/task has been completed, as such this provides an invaluable status for debugging and branching. Table 2 of Section 7 Nextion Return Data is not subject to bkcmd

14	rand	Random Value	n0.val=rand Readonly. Value returned by rand is random every time it is referred to. **default** range is 0 to 4294967295 range of rand is user customizable using the randset command range as set with randset will persist until reboot or reset
15	sys0 sys1 sys2	Numeric System Variables	sys0=10 sys1=40 sys2=60 n0.val=sys2 System Variables are global in nature with no need to define or create. They can be read or written from any page. 32-bit unsigned integers. min value of 0, max value of 4294967295 Suggested uses of sys variables include – as temporary variables in complex calculations – as parameters to pass to click function or pass between pages.
16	wup	Wake Up Page	wup=2, n0.val=wup Sets which page Nextion loads when exiting sleep mode min is 0, max is # of last page in HMI, or **default** 255 When wup=255 (not set to any existing page) – Nextion wakes up to current page, refreshing components only wup can be set even when Nextion is in sleep mode
17	usup	Wake On Serial Data	usup=0, usup=1 Sets if serial data wakes Nextion from sleep mode automatically. min is 0, max is 1, **default** 0 When usup=0, send sleep=0ÿÿÿ to wake Nextion When usup=1, any serial received wakes Nextion
18 *K*	rtc0 rtc1 rtc2 rtc3 rtc4 rtc5 rtc6	RTC	rtc0=2017, rtc1=8, rtc2=28, rtc3=16, rtc4=50, rtc5=36, n0.val=rtc6 Nextion RTC: rtc0 is year 2000 to 2099, rtc1 is month 1 to 12, rtc2 is day 1 to 31, rtc3 is hour 0 to 23, rtc4 is minute 0 to 59, rtc5 is second 0 to 59. rtc6 is dayofweek 0 to 6 (Sunday=0, Saturday=6) rtc6 is readonly and calculated by RTC when date is valid.
19 *K*	pio0 pio1 pio2 pio3 pio4 pio5 pio6 pio7	GPIO	pio3=1, pio3=0, n0.val=pio3 Default mode when power on: pull up input mode Internal pull up resistor: 50K GPIO is digital. Value of 0 or 1 only. – refer to cfgpio command for setting GPIO mode read if in input mode, write if in output mode

19	pwm4	PWM Duty	pwm4=25
K	pwm5	Cycle	Value in percentage. min 0, max 100, **default** 50.
	pwm6		– refer to cfgpio command for setting GPIO mode
	pwm7		
21	pwmf	PWM	pwmf=933
K		Frequency	Value is in Hz. min value 1 Hz, max value 65535 Hz. **default** 1000 Hz
			All PWM output is unified to only one Frequency, no independent individual settings are allowed.
			– refer to cfgpio command for setting GPIO mode
22	addr	Address	addr=257
			Advanced. Enables/disables Nextion's two byte Address Mode
			0, or min value 256, max value 2815. **default** 0
			Setting addr will persist to be the new power-on default.
			– refer to section 1.19
23	tch0	Touch	x.val=tch0, y.val=tch1
	tch1	Coordinates	Readonly. When Pressed tch0 is x coordinate, tch1 is y coordinate.
	tch2		When released (not currently pressed), tch0 and tch1 will be 0.
	tch3		tch2 holds the last x coordinate, tch3 holds the last y coordinate.
24	recmod	Protocol	recmod=0, recmod=1
		Reparse	**Advanced**. Set passive or active Protocol Reparse mode.
			min is 0, max is 1, **default** 0
			When recmod=0, Nextion is in passive mode and processes serial data according to the Nextion Instruction Set, this is the default power on processing. When recmod=1, Nextion enters into active mode where the serial data waits to be processed by event code. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.
25	usize	Bytes in	n0.val=usize
		Serial	**Advanced**. Read Only. Valid in active Protocol Reparse mode.
		Buffer	min is 0, max is 1024
			When Nextion is in active Protocol Reparse mode, usize reports the number of available bytes in the serial buffer. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.
26	u[index]	Serial	n0.val=u[0]
		Buffer Data	**Advanced**. Read Only. Valid in active Protocol Reparse mode.
			min is 0, max is 255
			When Nextion is in active Protocol Reparse mode, the u[index] array returns the byte at position index from the serial buffer. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.

7 – Format of Nextion Return Data

Return Codes dependent on bkcmd value being greater than 0

No.	Byte	bkcmd	len	Meaning	Format/Description
1	0x00	2,3	4	Invalid Instruction	0x00 0xFF 0xFF 0xFF Returned when instruction sent by user has failed
2	0x01	1,3	4	Instruction Successful	0x01 0xFF 0xFF 0xFF Returned when instruction sent by user was successful
3	0x02	2,3	4	Invalid Component ID	0x02 0xFF 0xFF 0xFF Returned when invalid Component ID or name was used
4	0x03	2,3	4	Invalid Page ID	0x03 0xFF 0xFF 0xFF Returned when invalid Page ID or name was used
5	0x04	2,3	4	Invalid Picture ID	0x04 0xFF 0xFF 0xFF Returned when invalid Picture ID was used
6	0x05	2,3	4	Invalid Font ID	0x05 0xFF 0xFF 0xFF Returned when invalid Font ID was used
7	0x11	2,3	4	Invalid Baud rate Setting	0x11 0xFF 0xFF 0xFF Returned when invalid Baud rate was used
8	0x12	2,3	4	Invalid Waveform ID or Channel #	0x12 0xFF 0xFF 0xFF Returned when invalid Waveform ID or Channel # was used
9	0x1A	2,3	4	Invalid Variable name or attribute	0x1A 0xFF 0xFF 0xFF Returned when invalid Variable name or invalid attribute was used
10	0x1B	2,3	4	Invalid Variable Operation	0x1B 0xFF 0xFF 0xFF Returned when Operation of Variable is invalid. ie: Text assignment t0.txt=abc or t0.txt=23, Numeric assignment j0.val=”50″ or j0.val=abc
11	0x1C	2,3	4	Assignment failed to assign	0x1C 0xFF 0xFF 0xFF Returned when attribute assignment failed to assign
12 *K*	0x1D	2,3	4	EEPROM Operation failed	0x1D 0xFF 0xFF 0xFF Returned when an EEPROM Operation has failed
13	0x1E	2,3	4	Invalid Quantity of Parameters	0x1E 0xFF 0xFF 0xFF Returned when the number of instruction parameters is invalid

14	0x1F	2,3	4	IO Operation failed	0x1F 0xFF 0xFF 0xFF Returned when an IO operation has failed
15	0x20	2,3	4	Escape Character Invalid	0x20 0xFF 0xFF 0xFF Returned when an unsupported escape character is used
16	0x23	2,3	4	Variable name too long	0x23 0xFF 0xFF 0xFF Returned when variable name is too long. Max length is 29 characters: 14 for page + "." + 14 for component.

Return Codes not affected by bkcmd value, valid in all cases

No.	Byte	length	Meaning	Format/Description
17	0x00	6	Nextion Startup	0x00 0x00 0x00 0xFF 0xFF 0xFF Returned when Nextion has started or reset
18	0x24	4	Serial Buffer Overflow	0x24 0xFF 0xFF 0xFF Returned when a Serial Buffer overflow occurs
19	0x65	7	Touch Event	0x65 0x00 0x01 0x01 0xFF 0xFF 0xFF Returned when Touch occurs and component's corresponding Send Component ID is checked in the users HMI design. 0x00 is page number, 0x01 is component ID, 0x01 is event (0x01 Press and 0x00 Release) **data**: Page 0, Component 1, Pressed
20	0x66	5	Current Page Number	0x66 0x01 0xFF 0xFF 0xFF Returned when the sendme command is used. 0x01 is current page number **data**: page 1
21	0x67	9	Touch Coordinate (awake)	0x67 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF Returned when sendxy=1 and not in sleep mode 0x00 0x7A is x coordinate in big endian order, 0x00 0x1E is y coordinate in big endian order, 0x01 is event (0x01 Press and 0x00 Release) (0x00*256+0x71,0x00*256+0x1E) **data**: (122,30) Pressed
22	0x68	9	Touch Coordinate (sleep)	0x68 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF Returned when sendxy=1 and in sleep mode 0x00 0x7A is x coordinate in big endian order, 0x00 0x1E is y coordinate in big endian order, 0x01 is event (0x01 Press and 0x00 Release) (0x00*256+0x71,0x00*256+0x1E) **data**: (122,30) Pressed

23	0x70	Varied	String Data Enclosed	0x70 0x61 0x62 0x31 0x32 0x33 0xFF 0xFF 0xFF Returned when using get command for a string. Each byte is converted to char. **data**: ab123
24	0x71	8	Numeric Data Enclosed	0x71 0x01 0x02 0x03 0x04 0xFF 0xFF 0xFF Returned when get command to return a number 4 byte 32-bit value in little endian order. (0x01+0x02*256+0x03*65536+0x04*16777216) **data**: 67305985
25	0x86	4	Auto Entered Sleep Mode	0x86 0xFF 0xFF 0xFF Returned when Nextion enters sleep automatically Using sleep=1 will not return an 0x86
26	0x87	4	Auto Wake from Sleep	0x87 0xFF 0xFF 0xFF Returned when Nextion leaves sleep automatically Using sleep=0 will not return an 0x87
27	0x88	4	Nextion Ready	0x88 0xFF 0xFF 0xFF Returned when Nextion has powered up and is now initialized successfully
28	0x89	4	Start microSD Upgrade	0x89 0xFF 0xFF 0xFF Returned when power on detects inserted microSD and begins Upgrade by microSD process
29	0xFD	4	Transparent Data Finished	0xFD 0xFF 0xFF 0xFF Returned when all requested bytes of Transparent Data mode have been received, and is now leaving transparent data mode (see 1.16)
30	0xFE	4	Transparent Data Ready	0xFE 0xFF 0xFF 0xFF Returned when requesting Transparent Data mode, and device is now ready to begin receiving the specified quantity of data (see 1.16)

1 – General Rules and Practices

No.	General Rule or Practice
1	**All instructions over serial: *are terminated with three bytes of* 0xFF 0xFF 0xFF** ie: decimal: 255 or hex: 0xFF or ansichar: ÿ or binary: 11111111 ie byte ndt[3] = {255,255,255}; write(ndt,3); or print("\xFF\xFF\xFF"); or print("ÿÿÿ")
2	All instructions and parameters are in ASCII
3	All instructions are in lowercase letters
4	Blocks of code and enclosed within braces { } can not be sent over serial this means if, for, and while commands can not be used over serial
5	A space char 0x20 is used to separate command from parameters.
6	There are no spaces in parameters unless specifically stated
7	Nextion uses integer math and does not have real or floating support.
8	Assignment are non-complex evaluating fully when reaching value after operator.
9	Comparison evaluation is non-complex. Use nesting when and as required.
10	Instructions over serial are processed on receiving termination (see 1.1)
11	Character escaping is limited to 0x0D by using two text chars \r
12	Nextion does not support order of operations. sys0=3+(8*4) is invalid.
13	16-bit 565 Colors are in decimal from 0 to 65535 (see 5.Note)
14	Text values must be encapsulated with double quotes: ie "Hello"
15	Items within specific to Enhanced Models are noted with ***K***

16	Transparent Data Mode (used by addt and wept commands)
	
	1. MCU sending to Nextion
	  1. MCU sends command. ie: wept 30,20ÿÿÿ or addt 1,0,320ÿÿÿ
	  2. Nextion requires ~5ms to prepare for transparent mode data transfer
	  3. Nextion sends "Ready" 0xFE 0xFF 0xFF 0xFF Return Data (see 7.30)
	  4. MCU can now send specified quantity (20) of raw bytes to Nextion
	  5. Nextion receives raw bytes from MCU until specified quantity (20) is received
	  6. Nextion sends "Finished" 0xFD 0xFF 0xFF 0xFF Return Data (see 7.29)
	  7. MCU and Nextion can proceed to next command
	
	Note: Nextion will remain waiting at step 5 until every byte of specified quantity is received.
	– During this time Nextion can not execute any other commands, and may indeed hang if the MCU fails to deliver the number of bytes as specified by the command parameter.
	– data quantity limited by serial buffer (all commands+terminations + data < 1024)

17	Only component attributes in green and non readonly system variables can be assigned new values at runtime. All others are readonly at runtime with the exception of .objname
18	Numeric values can now be entered with byte-aligned hex. ie: n0.val=0x01FF
19	**Advanced**. Address mode is an advanced technique prepending the serial instruction with two bytes for the address. Two byte address is to be sent in little endian order, ie: 2556 is sent 0xFC 0x09. By default, the Nextion address is 0 and does not require two byte prefixing. When the two

byte addressing is used, Nextion will only respond to the command if its address matches the two byte prefix, or the transmitted address is 65535 broadcast. See the addr system variable.

20 **Advanced**. Protocol Reparse mode is an advanced technique that allows users to define their own incoming protocol and incoming serial data handling. When in active Protocol Reparse mode, incoming serial data will not be processed natively by the Nextion firmware but will wait in the serial buffer for processing. To exit active Protocol Reparse mode, recmod must be set back to passive (ie: in Nextion logic as recmod=0), which can not be achieved via serial. Send DRAKJHSUYDGBNCJHGJKSHBDNÿÿÿ via serial to exit active mode serially. Most HMI applications will not require Protocol Reparse mode and should be skipped if not fully understood.

21 Commenting user code inside Events uses the double-slash (two characters of forward slash /) technique. See 2.27 for proper usage.

2 – Assignment Statements

No.	Data Type	Operator	Description/Example (see 1.8 and 1.17)
1	Text	=	Assignment. Right side will be evaluated with result placed in left side. Component .txt-maxl needs to be large enough to hold result. t0.txt=”Hello”
2	Text	+=	Text Addition. Will concatenate left side with right side with result placed left side. ie t0.txt+=”Hello” is equivalent to t0.txt=t0.txt+”Hello”. t0.txt=”-“+t0.txt becomes t0.txt=t0.txt+”-“. Use temp variable to prepend. va0.txt=t0.txt t0.txt=”-“+va0.txt t0.txt+=” World” // append ” World” to t0.txt //When contents of t0.txt is “Hello” becomes “Hello World”
3	Text	-=	Text Subtraction. Will remove right side (a specified numeric amount of characters to remove) from end of left side and result placed in left side. t0.txt-=4 or t0.txt=t0.txt-4 // remove last 4 chars from t0.txt
4	Text	\	Escape Character. Supported is \r hex 0x0D. (see 1.11) t0.txt=”\r”
5	Text	==	Boolean Equality. Evaluate left side to right side. If both left and right sides are the same creates a true condition if(t0.txt==va0.txt)
6	Text	!=	Boolean Inequality. Evaluate left side to right side. If both left and right sides are different creates a true condition if(t0.txt!=va0.txt)
7	Numeric	=	Assignment. Right side of equation will be evaluated and result placed in left side. If more than one operator on right side, full evaluation and assignment will occur at each operator. n0.val=bauds // places bauds value in n0.val component

8	Numeric	+=	Numeric Addition. Adds value of left side and right side with result placed in left side. n0.val+=4 is equivalent to n0.val=n0.val+4 n0.val+=va0.val
9	Numeric	-=	Numeric Subtraction. Subtracts right side from left side with result placed in left side. n0.val-=4 is equivalent to n0.val=n0.val-4 n0.val-=60 // decreases value of n0.val by 60
10	Numeric	*=	Numeric Multiplication. Multiplies left side with right side with product result placed in left side. n0.val*=2 is equivalent of n0.val=n0.val*2 n0.val*=2
11	Numeric	/=	Numeric Division. Equates division of numerator (left side) and divisor (right side) and places integer quotient in left side. 60000/20001=2 n0.val/=60
12	Numeric	%=	Numeric Modulo. Equates division of numerator (left side) and divisor (right side) and places integer remainder in left side. Divisor MUST be a constant. 60000/20001=19998 n0.val%=60
13	Numeric	<<	Arithmetic Bit Shift Left. Moves all bits specified number to the left. Using the 16-bit example that follows, (32-bit uses similar rules) All bits shifted above 15 are **lost** and **undefined** bits become 0 n0.val=n0.val<<4 **0 0 0 0.0 0 1 1.1 1 0 0.0 0 0 1** **0 0 1 1.1 1 0 0.0 0 0 1.** **0 0 1 1.1 1 0 0.0 0 0 1.0 0 0 0**
14	Numeric	>>	Arithmetic Bit Shift Right. Moves all bits specified number to the right. Using the 16-bit example that follows, (32-bit uses similar rules) All bits shifted below 0 are **lost** and **undefined** bits become the signed bit. When signed bit is 1 (value is negative) then left filled is with 1's When signed bit is 0 (value is positive) then left filled is with 0's n0.val=n0.val>>4 **0 0 0 0.0 0 1 1.1 1 0 0.0 0 0 1** **0 0 0 0.0 0 1 1.1 1 0 0** **0 0 0 0.0 0 0 0.0 0 1 1.1 1 0 0**
15	Numeric	&	Logical Bitwise AND. Compares all bits left side to all bits right side(mask) Using the 16-bit example that follows, (32-bit uses similar rules) Result is a bit of 1 where both left and right bits were 1 n0.val=n0.val&35381 **0 1 0 1.1 0 1 1.0 0 1 0.0 1 0 1** n0.val of 23333 **1 0 0 0.1 0 1 0.0 0 1 1.0 1 0 1** mask of 35381 **0 0 0 0.1 0 1 0.0 0 1 0.0 1 0 1** result is 2597

16	Numeric	|	Logical Bitwise OR. Compares all bits left side to all bits right side Using the 16-bit example that follows, (32-bit uses similar rules) Result is a bit of 1 where either left or right bits were 1 n0.val=n0.val|35381 **0 1 0 1.1 0 1 1.0 0 1 0.0 1 0 1** n0.val of 65519 **1 0 0 0.1 0 1 0.0 0 1 1.0 1 0 1** constant 35381 **1 1 0 1.1 0 1 1.0 0 1 1.0 1 0 1** result is 56117
17	Numeric	==	Boolean Equality. Evaluate left side to right side. If both left and right sides are the same creates a true condition if(n0.val==va0.val)
18	Numeric	!=	Boolean Inequality. Evaluate left side to right side. If both left and right sides are different creates a true condition if(n0.val!=va0.val)
19	Numeric	<	Boolean Less than. Evaluate left side to right side. If left side is less than right side creates a true condition while(n0.val<va0.val)
20	Numeric	<=	Boolean Less than or Equal. Evaluate left side to right side. If left side is less than or equal to right side creates a true condition while(n0.val<=va0.val)
21	Numeric	>	Boolean Greater than. Evaluate left side to right side. If left side is greater than right side creates a true condition while(n0.val>va0.val)
22	Numeric	>=	Boolean Greater than or Equal. Evaluate left side to right side. If left side is greater than or equal to right side creates a true condition while(n0.val>=va0.val)
23	Code	{ }	Code Block begins with open brace { on line by itself Code Block ends with closing brace } beginning a new line see if (see 3.25) while (see 3.26) and for (see 3.27)
24	Code	( )	Condition enclosure begins with open parenthesis (and ends with closing parenthesis) at end of line see if (see 3.25) while (see 3.26) and for (see 3.27)
25	Code	.	Period Separator. Separates Page, Component and Attributes Also used with page index and component array. (see 2.26) page1.va0.val or p0.pic
26	Code	[ ]	Array[index]. There are 3 arrays. Keyboard source showcases 2 arrays. The b[.id] component array which takes component .id as index The p[index] page array which takes page index as index These (p[].b[]) need to be used with caution and mindful purpose. Reference to a component without specified Attribute can create for long and potentially frustrating debug sessions. The third array is the Serial Buffer Data u[index]

			array. This is valid when in active Protocol Reparse mode. Protocol Reparse is an advanced technique that should be skipped if not fully understood. p[pageindex].b[component.id].attribute // global scope b[component.id].attribute // local scope on current page

27	Comment	//	Double-Slash Commenting to add user comments to code. Everything to the right of, and including, the double-slash is a comment that will not be executed by the Nextion interpreter. Comments should: 1) occur on a line by themselves with the double-slash at the beginning of the line (no leading spaces), 2) immediately following code on a line without a space separating code and the double slash. Commenting of code blocks should occur: 1) before the condition/iteration 2) inside the opening and closing braces 3) after the code block. *Notes:* It is important to note that comments can not interrupt code blocks without causing an “Error: Index was outside the bounds of the array”. Comments are counted towards the overall “code + attributes” hard limit of 65534.
			`// these are valid comments`
			`sys0=0// reset sys0 to zero`
			The following showcases valid/invalid use
			`//valid comment before condition/iteration`
			`for(sys0=0;sys0<=sys1;sys0++)`
			`// invalid comment between condition/iteration and block`
			`{`
			`  doevents//valid comment after code on same line`
			`  // valid comment inside block`
			`}`
			`// valid comment outside block`

3 – Operational Commands

No.	Name	Param Count	Description and Usage/Parameters/Examples
1	page	1	Change page to page specified. Unloads old page to load specified page. Nextion loads page 0 by default on power on. usage: page <pid> <pid> is either the page index number, or pagename page 0 // Change page to indexed page 0 page main // Change page to the page named main
2	ref	1	Refresh component (auto-refresh when attribute changes since v0.38) – if component is obstructed (stacking), ref brings component to top. usage: ref <cid> <cid> is component’s .id or .objname attribute of component to refresh – when <cid> is 0 (page component) refreshes all on the current page. ref t0 // Refreshes the component with .objname of t0 ref 3 // Refreshes the component with .id of 3 ref 0 // Refreshes all components on the current page (same as ref 255)

3	click	2	Trigger the specified components Touch Press/Release Event As event code is always local, object can not be page prefixed usage: click <cid>,<event> <cid> is component's .id or .objname attribute of component to refresh <event> is 1 to trigger Press Event, 0 to trigger Release Events click b0,1 // Trigger Touch Press Event of component with .objname b0 click 4,0 // Trigger Touch Release Event of component with .id 4
4	ref_stop	0	Stops default waveform refreshing (will not refresh when data point added) – waveform refreshing will resume with ref_star (see 3.5) usage: ref_stop ref_stop // stop refreshing the waveform on each data point added
5	ref_star	0	Resume default waveform refreshing (refresh on data point add) – used to resume waveform refreshing stopped by ref_stop (see 3.4) usage: ref_start ref_star // resume default refreshing, refresh on each data point added
6	get	1	Send attribute/constant over serial (0x70/0x71 Return Data) usage: get <attribute> <attribute> is either numeric value, .txt contents, or constant get t0.txt // sends text contents of t0.txt in 0x70 Return Data format get "123" // sends text constant "123" in 0x70 Return Data format get n0.val // sends numeric value of n0.val in 0x71 Return Data format get 123 // sends numeric constant 123 in 0x71 Return Data format
7	sendme	0	Sends number of currently loaded page over serial (0x66 Return Data) – number of currently loaded page is stored in system variable dp – used in a page's initialize event will auto-send as page loads usage: sendme sendme // sends the value of dp in 0x66 Return Data Format
8	cov	3	Convert variable from numeric type to text, or text to numeric type – text must be text ASCII representation of an integer value. – source and destination types must not be of the same type – when length is fixed and value is less, leading zeros will be added ie: src numeric value of 123 with length 4, result is dest text "0123" – dest txt_maxl and length needs be large enough to accommodate conversion. ie: src numeric value of 123 with length 0, result is dest text "123" – when value is larger than length, .txt **results in a truncation** – it is recommended to handle source length in user code before cov **Note:** v0.53 changed behaviour from previous pre v0.53 behaviours.

			cov is deemed undefined if source is larger than length or the dest txt_maxl is smaller than the requested length. Some undefines are exploitable. usage: cov <src>,<dest>,<length> <src> is text attribute (or numeric attribute when <dest> is text) <dest> is numeric attribute (or text attribute when <src> is numeric) <length> will determine if leading zeros added to conversion to text cov h0.val,t0.txt,0 // convert value of h0 into t0.txt without leading zeros cov t0.txt,h0.val,0 // convert integer into t0.txt from h0.val <length> ignored. cov h0.val,t0.txt,4 // convert value of h0 into t0.txt with exactly 4 digits Invalid: cov h0.val,va0.val,0 or cov t0.txt,va0.txt,0 // src & dest same type.
8a	covx	4	Convert variable from numeric type to text, or text to numeric type – text must be text ASCII representation of an integer value. – source and destination types must not be of the same type – when source is numeric, hex format and length not 0 and <4. ie: (len 2) positive significant (byte 0 to 3), 123 = 0000007B = 007B ie: (len 2) negative significant (byte 3 to 0), -123 = FFFFFF85 = FF85 – value is more than allowed space **results in a truncation** – it is recommended to ensure handling source length in user code before covx – in v0.53, covx is deemed undefined if source is larger than length or dest txt_maxl is smaller than requested length. (some of these undefines, can be exploited) ie: src numeric value of 123 with length 0, result is dest text “123” – when length is fixed and value is less, leading zeros will be added ie: src numeric value of 123 with length 4, result is dest text “0123” – when value is larger than length, .txt truncated to least significant digits ie: src numeric value of 23425 with length 4 result is dest text “3425” usage: covx <src>,<dest>,<length>,<format> <src> is text attribute (or numeric attribute when <dest> is text) <dest> is numeric attribute (or text attribute when <src> is numeric) <length> will determine if leading zeros added to conversion to text <format> 0: integer, 1: Comma separated 1,000s, 2: Hex covx h0.val,t0.txt,0,0 // convert value of h0 into t0.txt without leading zeros covx t0.txt,h0.val,0,0 // convert t0.txt into integer in h0.val <length> ignored. covx h0.val,t0.txt,4,0 // convert value of h0 into t0.txt with exactly 4 digits covx h0.val,t0.txt,4,1 // convert value of h0 into t0.txt with commas covx h0.val,t0.txt,4,2 // convert value of h0 into t0.txt in 2 bytes of hex digits Invalid: cov h0.val,va0.val,0 or cov t0.txt,va0.txt,0 // src & dest same type.
9	touch_j	0	Recalibrate the Nextion device’s touch sensor – presents 4 points on screen for user to touch, saves, and then reboots. – typically not required as device is calibrated at factory – sensor can be effected by changes of conditions in environment

			usage: touch_j
			touch_j // trigger the recalibration of touch sensor
10	substr	4	Extract character or characters from contents of a Text attribute
			usage: substr <src>,<dest>,<start>,<count>
			<src> is text attribute where text will be extracted from
			<dest> is text attribute to where extracted text will be placed
			<start> is start position for extraction (0 is first char, 1 second)
			<count> is the number of characters to extract
			substr va0.txt,t0.txt,0,5 // extract first 5 chars from va0.txt, put into t0.txt
11	vis	2	Hide or Show component on current page
			– show will refresh component and bring it to the forefront layer
			– hide will remove component visually, touch events will be disabled
			– use layering with mindful purpose, can cause ripping and flickering.
			– use with caution and mindful purpose, may lead to difficult debug session
			usage: vis <comp><state>
			<comp> is either .objname or .id of component on current page,
			– valid .id is 0 – page, 1 to 250 if component exists, and 255 for all
			<state> is either 0 to hide, or 1 to show.
			vis b0,0 // hide component with .objname b0
			vis b0,1 // show component with .objname b0, refresh on front layer
			vis 1,0 // hide component with .id 1
			vis 1,1 // show component with .id 1, refresh on front layer
			vis 255,0 // hides all components on the current page
12	tsw	2	Enable or disable touch events for component on current page
			– by default, all components are enabled unless disabled by tsw
			– use with caution and mindful purpose, may lead to difficult debug session
			usage: tsw <comp><state>
			<comp> is either .objname or .id of component on current page,
			– valid .id is 0 – page, 1 to 250 if component exists, and 255 for all
			<state> is either 0 to disable, or 1 to enable.
			tsw b0,0 // disable Touch Press/Release events for component b0
			tsw b0,1 // enable Touch Press/Release events for component b0
			tsw 1,0 // disable Touch Press/Release events for component with id 1
			tsw 1,1 // enable Touch Press/Release events for component with id 1
			tsw 255,0 // disable all Touch Press/Release events on current page
13	com_stop	0	Stop execution of instructions received from Serial
			– Serial will continue to receive and store in buffer.
			– execution of instructions from Serial will resume with com_star (see 3.14)
			– using com_stop and com_star may cause a buffer overflow condition.
			– Refer to device datasheet for buffer size and command queue size

			usage: com_stop
			com_stop // stops execution of instructions from Serial
14	com_star	0	Resume execution of instructions received from Serial
			– used to resume an execution stop caused by com_stop (see 3.13)
			– when com_star received, all instructions in buffer will be executed
			– using com_stop and com_star may cause a buffer overflow condition.
			– Refer to device datasheet for buffer size and command queue size
			usage: com_star
			com_star // resume execution of instruction from Serial
15	randset	2	Set the Random Generator Range for use with rand (see 6.14)
			– range will persist until changed or Nextion rebooted
			– set range to desired range before using rand
			– power on default range is -2147483648 to 2147483647, runtime range is user definable.
			usage: randset <min>,<max>
			<min> is value is -2147483648 to 2147483647
			<max> is value greater than min and less than 2147483647
			randset 1,100 //set current random generator range from 1 to 100
			randset 0,65535 //set current random generator range from 0 to 65535
16	code_c	0	Clear the commands/data queued in command buffer without execution
			usage: code_c
			code_c // Clears the command buffer without execution
17	print	1	Send raw formatted data over Serial to MCU
			– print/printh does not use Nextion Return Data, user must handle MCU side
			– qty of data may be limited by serial buffer (all data < 1024)
			– numeric value sent in 4 byte 32-bit little endian order
			value = byte1+byte2*256+byte3*65536+byte4*16777216
			– text content sent is sent 1 ASCII byte per character, without null byte.
			usage: print <attr>
			<attr> is either component attribute, variable or Constant
			print t0.txt // return 1 byte per char of t0.txt without null byte ending.
			print j0.val // return 4 bytes for j0.val in little endian order
			print “123” // return 3 bytes for text “123” 0x31 0x32 0x33
			print 123 // return 4 bytes for value 123 0x7B 0x00 0x00 0x00
17a	prints	2	Send raw formatted data over Serial to MCU
			– prints does not use Nextion Return Data, user must handle MCU side
			– qty of data may be limited by serial buffer (all data < 1024)
			– numeric value sent in 4 byte 32-bit little endian order
			value = byte1+byte2*256+byte3*65536+byte4*16777216
			– text content sent is sent 1 ASCII byte per character, without null byte.
			usage: prints <attr>,<length>

<attr> is either component attribute, variable or Constant

<length> is either 0 (all) or number to limit the bytes to send.

prints t0.txt,0 // return 1 byte per char of t0.txt without null byte ending.

prints t0.txt,4 // returns first 4 bytes, 1 byte per char of t0.txt without null byte ending.

prints j0.val,0 // return 4 bytes for j0.val in little endian order

prints j0.val,1 // returns 1 byte of j0.val in little endian order

prints “123”,2 // return 2 bytes for text “12” 0x31 0x32

prints 123,2 // returns 2 bytes for value 123 0x7B 0x00

18 printh 1 to many Send raw byte or multiple raw bytes over Serial to MCU

– printh is one of the few commands that parameter uses space char 0x20

– when more than one byte is being sent a space separates each byte

– byte is represented by 2 of (ASCII char of hexadecimal value per nibble)

– qty may be limited by serial buffer (all data < 1024)

– print/printh does not use Nextion Return Data, user must handle MCU side

usage: printh <hexhex>[<space><hexhex][…<space><hexhex]

<hexhex> is hexadecimal value of each nibble. 0x34 as 34

<space> is a space char 0x20, used to separate each <hexhex> pair

printh 0d // send single byte: value 13 hex: 0x0d

printh 0d 0a // send two bytes: value 13,10 hex: 0x0d0x0a

19 add 3 Add single value to Waveform Channel

– waveform channel data range is min 0, max 255

– 1 pixel column is used per data value added

– y placement is if value < s0.h then s0.y+s0.h-value, otherwise s0.y

usage: add <waveform>,<channel>,<value>

<waveform> is the .id of the waveform component

<channel> is the channel the data will be added to

<value> is ASCII text of data value, or numeric value

– valid: va0.val or sys0 or j0.val or 10

add 1,0,30 // add value 30 to Channel 0 of Waveform with .id 1

add 2,1,h0.val // add h0.val to Channel 1 of Waveform with .id 2

20 addt 3 Add specified number of bytes to Waveform Channel over Serial from MCU

– waveform channel data range is min 0, max 255

– 1 pixel column is used per data value added.

– addt uses Transparent Data Mode (see 1.16)

– waveform will not refresh until Transparent Data Mode completes.

– qty limited by serial buffer (all commands+terminations + data < 1024)

– also refer to add command (see 3.19)

usage: add <waveform>,<channel>,<qty>

<waveform> is the .id of the waveform component

<channel> is the channel the data will be added to

			<qty> is the number of byte values to add to <channel>
			addt 2,0,20 // adds 20 bytes to Channel 0 Waveform with .id 2 from serial
			// byte of data is not ASCII text of byte value, but raw byte of data.
21	cle	3	Clear waveform channel data
			usage: cle <waveform>,<channel>
			<waveform> is the .id of the waveform component
			<channel> is the channel to clear
			<channel> must be a valid channel: < waveform.ch or 255
			0 if .ch 1, 1 if .ch 2, 2 if .ch 3, 3 if .ch=4 and 255 (all channels)
			cle 1,0 // clear channel 0 data from waveform with .id 1
			cle 2,255 // clear all channels from waveform with .id 2
22	rest	0	Resets the Nextion Device
			usage: rest
			rest // immediate reset of Nextion device – reboot.
23	doevents	0	Force immediate screen refresh and receive serial bytes to buffer
			– useful inside exclusive code block for visual refresh (see 3.26 and 3.27)
			usage: doevents
			doevents // allows refresh and serial to receive during code block
24	strlen	2	Computes the length of string in <txt> and puts value in <len>
			usage: strlen <txt>,<len>
			<txt> must be a string attribute ie: t0.txt, va0.txt
			<len> must be numeric ie: n0.val, sys0, va0.val
			strlen t0.txt,n0.val // assigns n0.val with length of t0.txt content
24a	btlen	2	Computes number of bytes string in <txt> uses and puts value in <len>
			usage: btlen <txt>,<len>
			<txt> must be a string attribute ie: t0.txt, va0.txt
			<len> must be numeric ie: n0.val, sys0, va0.val
			btlen t0.txt,n0.val // assigns n0.val with number of bytes t0.txt occupies
25	if	Block	Conditionally execute code block if boolean condition is met
			– execute commands within block { } if (condition) is met.
			– nested conditions () is not allowed. invalid: ((h0.val+3)>0)
			– block opening brace { must be on line by itself
			– no space between block close brace } and else. valid: }else invalid: } else
			– Text comparison supports ==, !=
			– Numerical comparison supports >, <, ==, !=, >=, <=
			– nested “if” and “else if” supported.
			usage: if condition block [else if condition block] [else block]
			– (condition) is a simple non-complex boolean comparison evaluation
			valid: (j0.val>75) invalid: (j0.val+1>75) invalid: (j0.val<now.val+60)

```
if(t0.txt=="123456")
{
  page 1
}

if(n0.val==123)
{
  b0.txt="stop"
}else
{
  b0.txt="start"
}

if(rtc==1)
{
  t0.txt="Jan"
}else if(rtc1==2)
{
  t0.txt="Feb"
}else if(rtc1==3)
{
  t0.txt="Mar"
}else
{
  t0.txt="etc"
}
```

26 while Block Continually executes code block until boolean condition is no longer met

– tests boolean condition and execute commands within block { } if condition was met and continues to re-execute block until condition is not met.

– nested conditions () is not allowed. invalid: ((h0.val+3)>0)

– block opening brace { must be on line by itself

– Text comparison supports ==, !=

– Numerical comparison supports >, <, ==, !=, >=, <=

– block runs exclusively until completion unless doevents used (see 3.23)

usage: while condition block

– (condition) is a simple non-complex boolean comparison evaluation

valid: (j0.val>75) invalid: (j0.val+1>75)

```
// increment n0.val as lon as n0.val < 100.  result: b0.val=100
// will not visually see n0.val increment, refresh when while-loop completes
while(n0.val<100)
{
  n0.val++
}

//increment n0.val as long as n0.val < 100. result: n0.val=100
// will visually see n0.val increment, refresh each evaluation of while-loop
while(n0.val<100)
{
  n0.val++
  doevents
}
```

27 for Block Iterate execution of code block as long as boolean condition is met

– executes init_assignment, then tests boolean condition and executes

commands within block { } if boolean condition is met, when iteration of
block execution completes step_assignment is executed. Continues to
iterate block and step_assignment until boolean condition is not met.
– nested conditions () is not allowed. invalid: ((h0.val+3)>0)
– block opening brace { must be on line by itself
– Text comparison supports ==, !=
– Numerical comparison supports >, <, ==, !=, >=, <=
– block runs exclusively until completion unless doevents used (see 3.23)
usage: for(init_assignment;condition;step_assignment) block
– init_assignment and step_assignment are simple non-complex statement
valid: n0.val=4, sys2++, n0.val=sys2*4+3 invalid: n0.val=3+(sys2*4)-1
– (condition) is a simple non-complex boolean comparison evaluation
valid: (j0.val>75) invalid: (j0.val+1>75)

```
// iterate n0.val by 1's as long as n0.val<100. result: n0.val=100
// will not visually see n0val increment until for-loop completes
for(n0.val=0;n0.val<100;n0.val++)
{
}

////iterate n0.val by 2's as long as n0.val<100. result: n0.val=100
// will visually see n0.val increment when doevents processed
for(n0.val=0;n0.val<100;n0.val+=2)
{
  doevents
}
```

28 wepo 2 Store value/string to EEPROM

K

– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
– numeric value length: is 4 bytes, -2147483648 to 2147483647
– numeric data type signed long integer, stored in little endian order.
val[addr+3]*16777216+val[addr+2]*65536+val[addr+1]*256+val[addr]
– string content length: .txt content is .txt-maxl +1, or constant length +1
usage: wepo <attr>,<addr>
<attr> is variable or constant
<addr> is storage starting address in EEPROM
wepo t0.txt,10 // writes t0.txt contents in addresses 10 to 10+t0.txt-maxl
wepo “abcd”,10 // write constant “abcd” in addresses 10 to 14
wepo 11,10 // write constant 11 in addresses 10 to 13
wepo n0.val,10 // write value n0.val in addresses 10 to 13

29 repo 2 Read value from EEPROM

K

– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
– numeric value length: is 4 bytes, -2147483648 to 2147483647
– numeric data type signed long integer, stored in little endian order.
val[addr+3]*16777216+val[addr+2]*65536+val[addr+1]*256+val[addr]
– string content length: .txt content is lesser of .txt-maxl or until null reached.

			usage: repo <attr>,<addr>
			<attr> is variable or constant
			<addr> is storage starting address in EEPROM
			repo t0.txt,10 // reads qty .txt-maxl chars (or until null) from 10 into t0.txt
			repo n0.val,10 // reads 4 bytes from address 10 to 13 into n0.val
30	wept	2	Store specified number of bytes to EEPROM over Serial from MCU
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			– wept uses Transparent Data Mode (see 1.16)
			– qty limited by serial buffer (all commands+terminations + data < 1024)
			usage: wept <addr>,<qty>
			<addr> is storage starting address in EEPROM
			<qty> is the number of bytes to store
			wept 30,20 // writes 20 bytes into EEPROM addresses 30 to 49 from serial
			// byte of data is not ASCII text of byte value, but raw byte of data.
31	rept	2	Read specified number of bytes from EEPROM over Serial to MCU
K			– EEPROM valid address range is from 0 to 1023 (1K EEPROM)
			usage: rept <addr>,<qty>
			<addr> is storage starting address in EEPROM
			<qty> is the number of bytes to read
			rept 30,20 // sends 20 bytes from EEPROM addresses 30 to 49 to serial
			// byte of data is not ASCII text of byte value, but raw byte of data.
32	cfgpio	3	Configure Nextion GPIO
K			usage: cfgpio <io><mode><comp>
			<io> is the number of the extended I/O pin.
			– Valid values in PWM output mode: 4 to 7, all other modes 0 to 7.
			<mode> is the working mode of pin selected by <io>.
			– Valid values: 0-pull up input, 1-input binding, 2-push pull output,
			3-PWM output, 4-open drain output.
			<comp> component .objname or .id when <mode> is 1 (otherwise use 0)
			– in binding mode, cfgpio needs to be declared after every refresh of page to reconnect to
			Touch event, best to put cfgpio in page pre-initialization event
			cfgpio 0,0,0 // configures io0 as a pull-up input. Read as n0.val=pio0.
			cfgpio 1,2,0 // configures io1 as a push-pull output, write as pio1=1
			cfgpio 2,1, b0 // configures io2 as binding input with current page b0.
			// binding triggers b0 Press on falling edge and b0 Release on rising edge
			For PWM mode, set duty cycle before cfgpio: ie: pwm4=20 for 20% duty.
			cfgpio 4,3,0 // configures io4 as PWM output. pwmf=933 to change Hz.
			// changing pwmf changes frequency of all configured PWM io4 to io7
33	ucopy	4	**Advanced**. Read Only. Valid in active Protocol Reparse mode.
			Copies data from the serial buffer.

When Nextion is in active Protocol Reparse mode, ucopy copies data from the serial buffer. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.

usage: ucopy <attr>,<srcstart>,<len>,<deststart>

<attr> must be a writeable attribute ie: t0.txt, va0.val

<srcstart> must be numeric value ie: 0

<len> must be a numeric value ie: 4

<deststart> must be numeric value ie: 0

ucopy n0.val,0,2,0 // copy buffer bytes 0,1 to lower 2 bytes of n0.val

ucopy n0.val,0,2,2 // copy buffer bytes 0,1 to upper 2 bytes of n0.val

ucopy n0.val,0,4,0 // copy buffer bytes 0,1,2,4 to n0.val

ucopy t0.txt,0,10,0 // copy buffer bytes 0 to 9 into t0.txt

4 – GUI Designing Commands

No.	Name	Param Count	Description and Usage/Parameters/Examples
1	cls	1	Clear the screen and fill the entire screen with specified color usage: cls <color> <color> is either decimal 565 Color Value or Color Constant cls BLUE // Clear the screen and fill with color BLUE cls 1024 // Clear the screen and fill with color 1024
2	pic	3	Display a Resource Picture at specified coordinate usage: pic <x>,<y>,<picid> <x> is the x coordinate of upper left corner where picture should be drawn <y> is the y coordinate of upper left corner where picture should be drawn <picid> is the number of the Resource Picture in the HMI design pic 10,20,0 // Display Resource Picture #0 with upper left corner at (10,20) pic 40,50,1 // Display Resource Picture #1 with upper left corner at (40,50)
3	picq	5	Crop Picture area from Resource Picture using defined area – replaces defined area with content from the same area of Resource Picture – Resource Picture should be full screen-size or area might be undefined usage: picq <x>,<y>,<w>,<h>,<picid> <x> is the x coordinate of upper left corner of defined crop area <y> is the y coordinate of upper left corner of defined crop area <w> is the width of the defined crop area <h> is the height of the defined crop area <picid> is the number of the Resource Picture in the HMI design picq 20,50,30,20,0 // crops area 30×20, from (20,50) to (49,69), from Resource Picture 0

4	xpic	7	Advanced Crop Picture crop area from source Resource Picture render at destination coordinate usage: xpic <destx>,<desty>,<w>,<h>,<srcx>,<srcy>,<picid> <destx> is the x coordinate of destination upper left corner <desty> is the y coordinate of destination upper left corner <w> is the width of the defined crop area <h> is the height of the defined crop area <srcx> is the x coordinate of upper left corner of defined crop area <srcy> is the y coordinate of upper left corner of defined crop area <picid> is the number of the Resource Picture in the HMI design xpic 20,50,30,20,15,15,0 // crops area 30×20, from (15,15) to (44,34), // from Resource Picture 0 and renders it with upper left corner at (20,50)

5	xstr	11	Prints text on the Nextion device using defined area for text rendering
			usage: xstr <x>,<y>,<w>,<h>,<font>,<pco>,<bco>,<xcen>,<ycen>,<sta>,
			<text>
			<x> is the x coordinate of upper left corner of defined text area
			<y> is the y coordinate of upper left corner of defined text area
			<w> is the width of the defined text area
			<h> is the height of the defined text area
			<font> is the number of the Resource Font
			<pco> is the foreground color of text (Color Constant or 565 color value)
			<bco> is a) background color of text, or b) picid if <sta> is set to 0 or 2
			<xcen> is the Horizontal Alignment (0 – left, 1 – centered, 2 – right)
			<ycen> is the Vertical Alignment (0 – top/upper, 1 – center, 3 – bottom/lower)
			<sta> is background Fill (0 – crop image, 1 – solid color, 2 – image, 3 – none)
			<text> is the string content (constant or .txt attribute), ie “China”, or va0.txt
			xstr 10,10,100,30,1,WHITE,GREEN,1,1,1,va0.txt
			// use are 100×30 from (10,10) to (109,39) to print contents of va0.txt using
			// Font Resource 1 rendering Green letters on White background with both
			// horizontal and vertical centering and sta set as solid-color.

6	fill	5	Fill a defined area with specified color usage: fill <x>,<y>,<w>,<h>,<color> <x> is the x coordinate of upper left corner of defined fill area <y> is the y coordinate of upper left corner of defined fill area <w> is the width of the defined fill area <h> is the height of the defined fill area <color> is fill color, either decimal 565 Color Value or Color Constant fill 20,20,150,50,1024 // fills area 150×50 from (20,20) to (169,69) with 565 Color 1024.

7	line	5	Draw a line from point to point with specified color usage: line <x1>,<y1>,<x2>,<y2>,<color> <x1> is the x coordinate of the starting point of the line to be drawn <y1> is the y coordinate of the starting point of the line to be drawn <x2> is the x coordinate of the ending point of the line to be drawn <y2> is the y coordinate of the ending point of the line to be drawn <color> is line color, either decimal 565 Color Value or Color Constant line 20,30,170,200,BLUE // draws line in BLUE from (20,30) to (170,200)
8	draw	5	Draw a hollow rectangle around specified area with specified color usage: draw <x1>,<y1>,<x2>,<y2>,<color> <x1> is the x coordinate of the upper left corner of rectangle area <y1> is the y coordinate of the upper left corner of rectangle area <x2> is the x coordinate of the lower right corner of rectangle area <y2> is the y coordinate of the lower right corner of rectangle area <color> is line color, either decimal 565 Color Value or Color Constant draw 10,10,70,70,GREEN // draw a Green rectangle around (10,10) to (79,79) // effectively four lines from (x1,y1) to (x2,y1) to (x2,y2) to (x1,y2) to (x1,y1)
9	cir	4	Draw a hollow circle with specified radius and specified color usage: cir <x>,<y>,<radius>,<color> <x> is the x coordinate of the center point for the circle <y> is the y coordinate of the center point for the circle <radius> is the radius in pixels <color> is line color, either decimal 565 Color Value or Color Constant cir 100,100,30,RED // renders a hollow Red circle with circle center at (100,100), // a 30 pixel radius, a 61 pixel diameter, within boundary (70,70) to (130,130).
10	cirs	4	Draw a filled circle with specified radius and specified color usage: cirs <x>,<y>,<radius>,<color> <x> is the x coordinate of the center point for the circle <y> is the y coordinate of the center point for the circle <radius> is the radius in pixels <color> is fill color, either decimal 565 Color Value or Color Constant cir 100,100,30,RED // renders a filled Red circle with center at (100,100), // a 30 pixel radius, a 61 pixel diameter, within boundary (70,70) to (130,130).

5 – Color Code Constants

No.	Constant	565 Color Value	Indicator Color
1	BLACK	0	Black
2	BLUE	31	Blue
3	BROWN	48192	Brown
4	GREEN	2016	Green
5	YELLOW	65504	Yellow
6	RED	63488	Red
7	GRAY	33840	Gray
8	WHITE	65535	White

Note: 16-bit 565 Colors are in decimal values from 0 to 65535

24-bit RGB **11011000 11011000 11011000**

16-bit 565 **11011 110110 11011**

6 – System Variables

No.	Name	Meaning	Example/Description
1	dp	Current Page ID	dp=1, n0.val=dp read: Contains the current page displayed as per the HMI design write: change page to value specified (same effect as page command) min 0, max # of highest existing page in the user's HMI design.
2	dim dims	Nextion Backlight	dim=32, dims=100 Sets the backlight level in percent min 0, max 100, **default** 100 or user defined Note: dim=32 will set the current backlight level to 32%. using dims=32 will set the current backlight level to 32% and save this to be new power on default backlight level, persisting until changed.
3	baud bauds	Nextion Baud Rate	baud=9600, bauds=9600 Sets the Nextion Baud rate in bits-per-second min 2400, max 115200, **default** 9600 or user defined Valid values are: 2400, 4800, 9600, 19200, 38400, 57600, and 115200. Note: baud=38400 will set the current baud rate to 38400 using bauds=38400 will set the current baud rate to 38400 and save this to be new power on default baud rate, persisting until changed. Note: on rare occasions bauds has become lost. It is recommended to specify bauds=9600 in the first page's Preinitialization Event of HMI.
4	spax spay	Font Spacing	spax=2, spay=2 Globally sets the default rendering space: horizontally between font characters with spax additional pixels and vertically between rows (if multi-lined) with spay additional pixels. min 0, max 65535, **default** 0 Note: Components now have their own individual .spax/.spay attributes that are now used to determine spacing for the individual component.

5	thc	Touch Draw Brush Color	thc=RED, thc=1024 Sets the Touch Drawing brush color min 0, max 65535, **default** 0 Valid choices are either color constants or the decimal 565 color value.
6	thdra	Touch Drawing	thdra=1 (on), thdra=0 (off) Turns the internal drawing function on or off. min 0, max 1, **default** 0 When the drawing function is on, Nextion will follow touch dragging with the current brush color (as determined by the thc variable).
7	ussp	Sleep on No Serial	ussp=30 Sets internal No-serial-then-sleep timer to specified value in seconds min 3, max 65535, **default** 0 (max: 18 hours 12 minutes 15 seconds) Nextion will auto-enter sleep mode if and when this timer expires. Note: ussp=0 is an invalid value, meaning once ussp is set, it will persist and can not be unset unless through reboot or reset.
8	thsp	Sleep on No Touch	thsp=30 Sets internal No-touch-then-sleep timer to specified value in seconds min 3, max 65535, **default** 0 (max: 18 hours 12 minutes 15 seconds) Nextion will auto-enter sleep mode if and when this timer expires. Note: thsp=0 is an invalid value, meaning once thsp is set, it will persist and can not be unset unless through reboot or reset.
9	thup	Auto Wake on Touch	thup=0 (do not wake), thup=1 (wake on touch) Sets if Nextion should auto-wake from sleep when touch press occurs. min 0, max 1, **default** 0 When value is 1 and Nextion is in sleep mode, the first touch will only trigger the auto wake mode and not trigger a Touch Event. thup has no influence on sendxy, sendxy will operate independently.
10	sendxy	RealTime Touch Coordinates	sendxy=1 (start sending) sendxy=0 (stop sending) Sets if Nextion should send 0x67 and 0x68 Return Data min 0, max 1, **default** 0 – Less accurate closer to edges, and more accurate closer to center. Note: expecting exact pixel (0,0) or (799,479) is simply not achievable.
11	delay	Delay	delay=100 Creates a halt in Nextion code execution for specified time in ms min 0, max 65535 As delay is interpreted, a total halt is avoided. Incoming serial data is received and stored in buffer but not be processed until delay ends. If delay of more than 65.535 seconds is required, use of multiple delay statements required. delay=-1 is max. 65.535 seconds.

12	sleep	Sleep	sleep=1 (Enter sleep mode) or sleep=0 (Exit sleep mode) Sets Nextion mode between sleep and awake. min 0, max 1, or **default** 0 When exiting sleep mode, the Nextion device will auto refresh the page (as determined by the value in the wup variable) and reset the backlight brightness (as determined by the value in the dim variable).
13	bkcmd	Pass / Fail Return Data	bkcmd=3 Sets the level of Return Data on commands processed over Serial. min 0, max 3, **default** 2 – Level 0 is Off – no pass/fail will be returned – Level 1 is OnSuccess, only when last serial command successful. – Level 2 is OnFailure, only when last serial command failed – Level 3 is Always, returns 0x00 to 0x23 result of serial command. Result is only sent after serial command/task has been completed, as such this provides an invaluable status for debugging and branching. Table 2 of Section 7 Nextion Return Data is not subject to bkcmd
14	rand	Random Value	n0.val=rand Readonly. Value returned by rand is random every time it is referred to. **default** range is 0 to 4294967295 range of rand is user customizable using the randset command range as set with randset will persist until reboot or reset
15	sys0 sys1 sys2	Numeric System Variables	sys0=10 sys1=40 sys2=60 n0.val=sys2 System Variables are global in nature with no need to define or create. They can be read or written from any page. 32-bit unsigned integers. min value of 0, max value of 4294967295 Suggested uses of sys variables include – as temporary variables in complex calculations – as parameters to pass to click function or pass between pages.
16	wup	Wake Up Page	wup=2, n0.val=wup Sets which page Nextion loads when exiting sleep mode min is 0, max is # of last page in HMI, or **default** 255 When wup=255 (not set to any existing page) – Nextion wakes up to current page, refreshing components only wup can be set even when Nextion is in sleep mode
17	usup	Wake On Serial Data	usup=0, usup=1 Sets if serial data wakes Nextion from sleep mode automatically. min is 0, max is 1, **default** 0 When usup=0, send sleep=0ÿÿÿ to wake Nextion When usup=1, any serial received wakes Nextion

18	rtc0	RTC	rtc0=2017, rtc1=8, rtc2=28,
K	rtc1		rtc3=16, rtc4=50, rtc5=36, n0.val=rtc6
	rtc2		Nextion RTC:
	rtc3		rtc0 is year 2000 to 2099, rtc1 is month 1 to 12, rtc2 is day 1 to 31,
	rtc4		rtc3 is hour 0 to 23, rtc4 is minute 0 to 59, rtc5 is second 0 to 59.
	rtc5		rtc6 is dayofweek 0 to 6 (Sunday=0, Saturday=6)
	rtc6		rtc6 is readonly and calculated by RTC when date is valid.
19	pio0	GPIO	pio3=1, pio3=0, n0.val=pio3
K	pio1		Default mode when power on: pull up input mode
	pio2		Internal pull up resistor: 50K
	pio3		GPIO is digital. Value of 0 or 1 only.
	pio4		– refer to cfgpio command for setting GPIO mode
	pio5		read if in input mode, write if in output mode
	pio6		
	pio7		
19	pwm4	PWM Duty	pwm4=25
K	pwm5	Cycle	Value in percentage. min 0, max 100, **default** 50.
	pwm6		– refer to cfgpio command for setting GPIO mode
	pwm7		
21	pwmf	PWM	pwmf=933
K		Frequency	Value is in Hz. min value 1 Hz, max value 65535 Hz. **default** 1000 Hz
			All PWM output is unified to only one Frequency, no independent individual settings are allowed.
			– refer to cfgpio command for setting GPIO mode
22	addr	Address	addr=257
			Advanced. Enables/disables Nextion's two byte Address Mode
			0, or min value 256, max value 2815. **default** 0
			Setting addr will persist to be the new power-on default.
			– refer to section 1.19
23	tch0	Touch	x.val=tch0, y.val=tch1
	tch1	Coordinates	Readonly. When Pressed tch0 is x coordinate, tch1 is y coordinate.
	tch2		When released (not currently pressed), tch0 and tch1 will be 0.
	tch3		tch2 holds the last x coordinate, tch3 holds the last y coordinate.
24	recmod	Protocol	recmod=0, recmod=1
		Reparse	**Advanced**. Set passive or active Protocol Reparse mode.
			min is 0, max is 1, **default** 0
			When recmod=0, Nextion is in passive mode and processes serial data according to the Nextion Instruction Set, this is the default power on processing. When recmod=1, Nextion enters into active mode where the serial data waits to be processed by event code. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.

25	usize	Bytes in Serial Buffer	n0.val=usize **Advanced**. Read Only. Valid in active Protocol Reparse mode. min is 0, max is 1024 When Nextion is in active Protocol Reparse mode, usize reports the number of available bytes in the serial buffer. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.
26	u[index]	Serial Buffer Data	n0.val=u[0] **Advanced**. Read Only. Valid in active Protocol Reparse mode. min is 0, max is 255 When Nextion is in active Protocol Reparse mode, the u[index] array returns the byte at position index from the serial buffer. Most HMI applications will not require Protocol Reparse and should be skipped if not fully understood.

7 – Format of Nextion Return Data

Return Codes dependent on bkcmd value being greater than 0

No.	Byte	bkcmd	len	Meaning	Format/Description
1	0x00	2,3	4	Invalid Instruction	0x00 0xFF 0xFF 0xFF Returned when instruction sent by user has failed
2	0x01	1,3	4	Instruction Successful	0x01 0xFF 0xFF 0xFF Returned when instruction sent by user was successful
3	0x02	2,3	4	Invalid Component ID	0x02 0xFF 0xFF 0xFF Returned when invalid Component ID or name was used
4	0x03	2,3	4	Invalid Page ID	0x03 0xFF 0xFF 0xFF Returned when invalid Page ID or name was used
5	0x04	2,3	4	Invalid Picture ID	0x04 0xFF 0xFF 0xFF Returned when invalid Picture ID was used
6	0x05	2,3	4	Invalid Font ID	0x05 0xFF 0xFF 0xFF Returned when invalid Font ID was used
7	0x11	2,3	4	Invalid Baud rate Setting	0x11 0xFF 0xFF 0xFF Returned when invalid Baud rate was used
8	0x12	2,3	4	Invalid Waveform ID or Channel #	0x12 0xFF 0xFF 0xFF Returned when invalid Waveform ID or Channel # was used
9	0x1A	2,3	4	Invalid Variable name or attribute	0x1A 0xFF 0xFF 0xFF Returned when invalid Variable name or invalid attribute was used

10	0x1B	2,3	4	Invalid Variable Operation	0x1B 0xFF 0xFF 0xFF Returned when Operation of Variable is invalid. ie: Text assignment t0.txt=abc or t0.txt=23, Numeric assignment j0.val=”50″ or j0.val=abc
11	0x1C	2,3	4	Assignment failed to assign	0x1C 0xFF 0xFF 0xFF Returned when attribute assignment failed to assign
12 *K*	0x1D	2,3	4	EEPROM Operation failed	0x1D 0xFF 0xFF 0xFF Returned when an EEPROM Operation has failed
13	0x1E	2,3	4	Invalid Quantity of Parameters	0x1E 0xFF 0xFF 0xFF Returned when the number of instruction parameters is invalid
14	0x1F	2,3	4	IO Operation failed	0x1F 0xFF 0xFF 0xFF Returned when an IO operation has failed
15	0x20	2,3	4	Escape Character Invalid	0x20 0xFF 0xFF 0xFF Returned when an unsupported escape character is used
16	0x23	2,3	4	Variable name too long	0x23 0xFF 0xFF 0xFF Returned when variable name is too long. Max length is 29 characters: 14 for page + “.” + 14 for component.

Return Codes not affected by bkcmd value, valid in all cases

No.	Byte	length	Meaning	Format/Description
17	0x00	6	Nextion Startup	0x00 0x00 0x00 0xFF 0xFF 0xFF Returned when Nextion has started or reset
18	0x24	4	Serial Buffer Overflow	0x24 0xFF 0xFF 0xFF Returned when a Serial Buffer overflow occurs
19	0x65	7	Touch Event	0x65 0x00 0x01 0x01 0xFF 0xFF 0xFF Returned when Touch occurs and component’s corresponding Send Component ID is checked in the users HMI design. 0x00 is page number, 0x01 is component ID, 0x01 is event (0x01 Press and 0x00 Release) **data**: Page 0, Component 1, Pressed
20	0x66	5	Current Page Number	0x66 0x01 0xFF 0xFF 0xFF Returned when the sendme command is used. 0x01 is current page number **data**: page 1

21	0x67	9	Touch Coordinate (awake)	0x67 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF Returned when sendxy=1 and not in sleep mode 0x00 0x7A is x coordinate in big endian order, 0x00 0x1E is y coordinate in big endian order, 0x01 is event (0x01 Press and 0x00 Release) (0x00*256+0x71,0x00*256+0x1E) **data**: (122,30) Pressed
22	0x68	9	Touch Coordinate (sleep)	0x68 0x00 0x7A 0x00 0x1E 0x01 0xFF 0xFF 0xFF Returned when sendxy=1 and in sleep mode 0x00 0x7A is x coordinate in big endian order, 0x00 0x1E is y coordinate in big endian order, 0x01 is event (0x01 Press and 0x00 Release) (0x00*256+0x71,0x00*256+0x1E) **data**: (122,30) Pressed
23	0x70	Varied	String Data Enclosed	0x70 0x61 0x62 0x31 0x32 0x33 0xFF 0xFF 0xFF Returned when using get command for a string. Each byte is converted to char. **data**: ab123
24	0x71	8	Numeric Data Enclosed	0x71 0x01 0x02 0x03 0x04 0xFF 0xFF 0xFF Returned when get command to return a number 4 byte 32-bit value in little endian order. (0x01+0x02*256+0x03*65536+0x04*16777216) **data**: 67305985
25	0x86	4	Auto Entered Sleep Mode	0x86 0xFF 0xFF 0xFF Returned when Nextion enters sleep automatically Using sleep=1 will not return an 0x86
26	0x87	4	Auto Wake from Sleep	0x87 0xFF 0xFF 0xFF Returned when Nextion leaves sleep automatically Using sleep=0 will not return an 0x87
27	0x88	4	Nextion Ready	0x88 0xFF 0xFF 0xFF Returned when Nextion has powered up and is now initialized successfully
28	0x89	4	Start microSD Upgrade	0x89 0xFF 0xFF 0xFF Returned when power on detects inserted microSD and begins Upgrade by microSD process
29	0xFD	4	Transparent Data Finished	0xFD 0xFF 0xFF 0xFF Returned when all requested bytes of Transparent Data mode have been received, and is now leaving transparent data mode (see 1.16)
30	0xFE	4	Transparent Data Ready	0xFE 0xFF 0xFF 0xFF Returned when requesting Transparent Data mode, and device is now ready to begin receiving the specified quantity of data (see 1.16)