The Nextion Instruction Set

There are that all instructions in one.
They categorized into only a few categories:

General Rules and Practices ... ~~gray~~
Operation Commands ... ~~gray~~
Operational Commands ... ~~gray~~
Variable Commands ... ~~gray~~
Data Cache Commands ... ~~gray~~
System Variables ... ~~gray~~
Serial or Nextion Macro Data ... ~~gray~~

Legend:

Bold	Defaults
Italic	Optional
Underline	Tip
Red	Unused
Gray	Deprecated

1 – General Rules and Practices

No.	General Rule or Practice
1	**All instructions over serial are terminated with three bytes of hex ff ff ff**
	or decimal 255 or less (left of another 1) or binary 11111111
	In byte HEX(255),CHR(255),ASCII(255),or prompt("ctrl+alt+ff"),or prompt("ctrl+alt+255")
2	All instructions and parameters are in ASCII
3	All instructions are in lowercase letters
4	Checks of code and executed within hexes (1) can not be sent over serial
	Hex macro & file, and while commands can not be used over serial.
5	Nextion char 0x00 is used to separate command from parameters.
6	Back to operation in parameters unless specifically stated
7	Nextion uses integer math and does not have real or floating support.
8	Nextion can non-complex evaluate fully while reaching value after operation
9	Comparison evaluation is non-complex, but can be joined (see && and ||)
10	Nextion does not process and unprocessed an incoming Wrtmodifiers field
11	Nextion escaping is performed using two bytes: 0 creates 1 Byte 0x00-0xbb, 7 HLC and 1 Tx 0xbc
12	Nextion does not support order of operations, specifically OPS is needed
13	0 HEX bits Colors are decimal from 0 to 65535 (see **5.Color**)
14	Not all pages must be recognized with Nextion Macro Data (see **5.Macro**)
15	Nextion which are specific to the Nextion Macro Data (see **5.Macro**)
16	Transparent Data Mode (see **5.Data** and **6.Data**)
	MCU sending to Nextion
	MCU sends instead: 0x aa 00,200,0 or aa01,5,0,255,0
	Nextion response: Ceiss ok prepare for transparent mode data transfer
	Nextion sends "Ready" 0x07 0x07 0x07 Return Data (see **5.Data**)
	MCU can now send specified quantity (200 of aa bytes in Nextion
	Nextion receives specified quantity (200 of aa bytes) and sends "Ceiss ok"
	Nextion sends "Finished" 0x07 0x07 0x07 Return Data (see **5.Data**)
	MCU and Nextion are prepared to resume normal
	Note: Nextion will remain waiting at step 5 until every byte of specified quantity is received,
	- During this Nextion Data transfer all other commands and requests are ignored. Using if the MCU fails to deliver the number of bytes as specified by the instructions
	- data quantity limited by serial buffer (all commands/instructions + data < 1024)
17	If ctrl+ component attributes in grand and non-readonly system variables can be assigned new values at runtime,
	the values are in nonlinear units with the exception of .pkaction
18	Numeric values can be entered with byte aligned hex, in ascii (0x41 or 0x61)
19	**Escaped**: Nextion reader is an advanced technique **Escaping** the end instruction with two bytes for the address. Two bytes address to be sent in little endian order; to 255e is sent 0xff, 0x0e(all, the Nextion address is not required when two bytes preceding the end instruction are 0x00 and 0x00 (which is required to the command if the address matches the two byte prefix, or the transmitted address is HEX(0xxxxxxx, 0x00, 0x00) or HEX(0xxxxxxx, 0x00, 0x00, 0x00))
20	**Escaped**: Nextion reader is an advanced technique **Escaping** the end instruction with two bytes for the address. Two bytes address to be sent in little endian order; to 255e is sent 0xff, 0x0e(all, the Nextion address is not required when two preceding the end instruction are 0x00 and 0x00 (which is required to the command if the address matches the two byte prefix, or the transmitted address is HEX(0xxxxxxx, 0x00, 0x00) or HEX(0xxxxxxx, 0x00, 0x00, 0x00))
21	Nextion Serial Protocol Reduce (NSPR) is an advanced technique that allows sending commands over serial protocol and receiving serial data. Similarly to active Protocol Response mode, incoming serial data will not be processed by the Nextion firmware but will be sent to Nextion Serial Protocol Reduce mode. Nextion Serial Protocol Reduce mode, then must be set back to passive or to Nextion Reader, as necessary, which can not be accomplished by sending the command: DBAUD=DBAUD+PASS(DBAUD);,as serial is set back to DBAUD. Applications will not require Protocol Response mode and should be skipped if not fully understood.
22	Processing over code blocks over 1024 characters causes the double stack bass characters of forward stack ( ) textbraqc. See **2.80** for proper usage.

2 – Assignment Statements

No.	Data Type	Operation	Description/Examples See **5.0** and **5.01**
1	Text	=	Assignment. Right side will be evaluated with result placed in left side. Component
			attributes can be set large enough to hold result.
			vtxt="Hello"
2	Text	+	Text Addition. Will concatenate left side with right side with result placed left side. In
			vtxt="Hello" is equivalent to vtxt=vtxt+"World"
			t0.txt="Hello" is equivalent to t0.txt=t0.txt+"World"
			Use temp variable to prepend: vtxt=vtxt+"Hello" is equivalent to
			t0.txt="World" if prepend "World" to t0.txt
			Integer conversion of ID is "Hello" becomes "Hello World"
3	Text	-	Text Subtraction. Will remove right side text in specified numeric amount of characters to
			removed from end of left side and result placed in left side.
			vtxt=vtxt-10 vtxt=vtxt-t0.n removes last 10 chars from vtxt
4	Text	*	Text Character Generator. Uses ASII
			Supported is 1 hex 0x00-0xbb, 7 hex 0x01, 0 hex 0xbc,
			0xbc="2"
5	Text	*	Boolean Equality. Evaluate left side to right side.
			Both left and right sides are the same creates a true condition
			vtxt0==vtxt1
6	Text	*	Boolean Inequality. Evaluate left side to right side.
			If both left and right sides are different creates a true condition
			vtxt0!=vtxt1
7	Numeric	=	Assignment. Right side of equation will be evaluated and result placed in left side. If
			Less than one operator on right side, full evaluation and assignment will occur at runtime
			n0=n1
			n0=n1+n2 (Calculate Sum) evaluate n1+n2 component
8	Numeric	+	Numeric Addition. Adds value of left side and right side with result placed in left side.
			n0=n0+n1 is equivalent to n0=n0+n1
			n0=n0+n1 decreases value of n0 by n1
9	Numeric	-	Numeric Subtraction. Subtracts right side from left side with result placed in left side.
			n0=n0-n1 is equivalent to n0=n0-n1
			n0=n0-n1 decreases value of n0 by n1
10	Numeric	*	Numeric Multiplication. Multiplies left side and right side with product result placed
			in left side. n0=n0*n1 is equivalent of n0=n0+n1
			n0=n0*n1
11	Numeric	/	Numeric Division. Equates division of numerator (left side) and divisor (right side) and
			places integer numerator in left side. 6000/1000=6
			n0=n0/n0
12	Numeric	%	Numeric Modulus. Equates division of numerator (left side) and divisor (right side) and
			places integer remainder in left side. Divisor MUST be a constant. 6000/2000=0999
			n0=n0%n0
13	Numeric	^	Arithmetic Bit Shift Left. Moves all bits specified number to the left.
			Using the 16 bit example that follows, 210 bit uses lowest bits
			All bits shifted below 0 are lost and **undefined** bits become the signed bit.
			When signed bit is 1 below is negated then left filled is zero.
			When signed bit is 0 below is positive then left filled is zero
			n0=n0<<n0=n1
			0000000011110000
			0000000000111100
			0000000000111100
14	Numeric	^	Arithmetic Bit Shift Right. Moves all bits specified number to the right.
			Using the 16 bit example that follows, 210 bit uses lowest bits
			All bits shifted below 0 are lost and **undefined** bits become the signed bit.
			When signed bit is 1 below is negated then left filled is zero.
			When signed bit is 0 below is positive then left filled is zero
			n0=n0>>n0=n1
			0000000011110000
			0000000000111100
			0000000000111100
15	Numeric	&	Logical Binary AND. Evaluates all bits left side to all bits right side
			Using the 16 bit example that follows, 210 bit uses similar hexes
			Result is a bit of 1 where both left and right bits were 1
			n0=n0&n0=n1
			1111111111111111
			1111111111111111
			1111111111111111
16	Numeric	|	Logical Binary OR. Evaluates all bits left side to all bits right side
			Using the 16 bit example that follows, 210 bit uses similar hexes
			Result is a bit of 1 where either left or right bits were 1
			n0=n0|n0=n1
			1111111111111111
			1111111111111111
			1111111111111111
17	Numeric	~	Logical Binary XOR. Applies bit inversion to all bits in the bitmask
			Using the 16 bit example that follows, 210 bit uses similar hexes
			Result is a bit inverted where mask0=0 was 1, unchanged where mask0=0 was 0
			n0=n0~n0=n1
			0111111111111110
			1111111111111111
			1000000000000001
18	Numeric	!	Boolean Equality. Evaluate left side to right side.
			If both left and right sides are the same creates a true condition
			n0!=n0
19	Numeric	!	Boolean Inequality. Evaluate left side to right side.
			If both left and right sides are different creates a true condition
			n0!=n0
20	Numeric	>	Boolean Less than. Evaluate left side to right side.
			If left side is less than right side creates a true condition
			n0>n0=n0
21	Numeric	<	Boolean Greater than. Evaluate left side to right side.
			If left side is greater than right side creates a true condition
			n0<n0=n0
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
			n0>=n0=n0
23	Numeric	<=	Code Block begins with open brace (on line by itself
			Code block ends with closing brace) beginning on new line
			on (line $0, n0=n1 **n0** and for line $0,
24	Code	?:	Condition (else-if)
			Conditions may be joined with no space between conditions, left to right evaluation
			on (line $0 **n0** and for line $0
25	Code	?	Condition (else-OR
			Conditions may be joined with no space between conditions, left to right evaluation
			on (line $0 **n0** and for line $0
26	Code	&&	Condition (and-AND
			Conditions may be joined with no space between conditions, left to right evaluation
			on (line $0 **n0** and for line $0
27	Code	||	Code block begins with open parenthesis (
			Code block ends with closing parenthesis) beginning on new line
			Code block can be nested to create complex statements.
			on (line $0 **n0** and for line $0
28	Code	(	Percent Separator. Separates Page, Component and Attributes.
			Use used with page index and component name to access the
			page0.vtxt or t0.pic
29	Code	%	Page[Num]. There are 1 arrays, keyboard source characters 2 arrays,
			The 5[Left] component array which takes component id as index
			The p[Index] page array which takes page index as index
			These [p][5] need to be used with caution and minefuls purpose. Reference to a
			component without specified attribution can create for loop and unpredictable, frustrating,
			debug sessions. The third array is the Serial Buffer Data p[Index].serial. This is well when
			to access Protocol Response mode. Protocol Response is an advanced technique that should
			be stripped if not fully understood.
			p[pageindex][component].attribute // global scope
			component[].attribute // local scope or current page
30	Comment		Double Slash Commenting out with user comments is a
			everything to the right, and including, the double slash
			Commenting out code blocks, which are code that will occur in a line by themselves
			with the double slash at the beginning of the line, is well suited for debugging.
			following code on a line without a space separating code and the double slash.
			Commenting of code blocks should occur 1 Byte before the conditions/function if inside the
			opening and closing braces. If after the code block, Brace. 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 = attribute" hard
			limit of 1024.
			// Show serial buffer
			tsdbhhi vtxt0 page 0a prev
			The following showcases vxtr0/vxtr0 use
			//called comment before condition/function
			//inside code block comment
			// invalid comment between condition/function and block
			if(vxtr0==0)
			//executed/called comment after code on same line
			// serial comment inside block
			// called comment outside block
31	Code		Keyboard. There are 2 arrays, keyboard source characters 2 arrays,
			The k[Left] component array which takes component id as index
			The p[Index] page array which takes page index as index
			These [p][5] need to be used with caution and minefuls purpose. Reference to a
			component without specified attribution can create for loop and unpredictable, frustrating,
			debug sessions. The third array is the Serial Buffer Data p[Index].serial. This is well when
			to access Protocol Response mode. Protocol Response is an advanced technique that should
			be stripped if not fully understood.
			p[pageindex][component].attribute // global scope
			component[].attribute // local scope or current page
32	Comment		Double Slash Commenting out with user comments is a
			Everything to the right, and including, the double slash
			Commenting out code blocks, which are code that will occur in a line by themselves
			with the double slash at the beginning of the line, is well suited for debugging.
			following code on a line without a space separating code and the double slash.
			Commenting of code blocks should occur 1 Byte before the conditions/function if inside the
			opening and closing braces. If after the code block, Brace. 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 = attribute" hard
			limit of 1024.
			// Show serial buffer
			tsdbhhi vtxt0 page 0a prev
			The following showcases vxtr0/vxtr0 use
			//called comment before condition/function
			//inside code block comment
			// invalid comment between condition/function and block
			if(vxtr0==0)
			//executed/called comment after code on same line
			// serial comment inside block
			// called comment outside block

3 – Operational Commands

No.	Command	Parameters	Description and Usage/Parameters/Examples
	Command	Defaults	
1	page		Change page to page specified. Unleash not page to load specified page.
			Nextion loads page 0 by default on power on.
			page 0,1,2,3,4,5,6,7,8,9
			<pstr> is either the page index number or pagename
			page 0,1 Change page to indexed page 1
			page main,1 Change page to the page named main
2	ref		Refresh component (page0.name) refresh when attribute changes since ab.00
			component is abbreviated/shortcut/key-of-things component to be
			usage: ref <cn>
			<cn> is component1.id or pkaction attribute of component to refresh
			<when> <cn> is 0 (page component) refreshes all on the current page.
			ref 0,2 Refreshes the component with .pkaction id of 2
			ref t1,2 Refreshes the component with .id of 2
			ref 0,2 Refreshes all components on the current page (same as ref 255)
3	click		Trigger the specified component Touch Press/Release Event
			<event code> is always 1 (touch down) or 0 (touch up)
			usage: click <cn> <event>
			<cn> is component1.id or pkaction attribute of component to refresh
			<event> is 1 or higher Press Event, 0 or higher Release Event
			click t0,1,1 Trigger Touch Press Event of component with id of 0
			click t0,1,0 Trigger Touch Release Event of component with id of 0
4	ref_stop		Stops default unwaitlisted refreshing (set not refresh when data point added)
			unwaitlisted refreshing will resume after refresh command
			usage: ref_stop
			ref_stop,1 stop refreshing the waitlisted on each data point added
5	ref_star		Resume default waitlisted refreshing (refresh on data point added)
			used to resume waitlisted refreshing stopped by ref_stop command
			usage: ref_start
			ref_star,1 resume default refreshing, refresh on each data point added
6	get		Send attribute/constant over serial (0x55/0x71 Return Data)
			usage: get <attribute>
			<attribute> is either numeric value, int constants, or component
			get 0,0x41 Sends text contents of Strict in 0x71 Return Data format
			"get 0" Sends text contents "0" in 0x70 Return Data format
			get vtxt0 Sends numeric value of vtxt0 in 0x70 Return Data format
			get t32 Sends numeric value0 t32 in 0x70 Return Data format
7	sendme		Sends number of currently loaded page over serial (0x66 Return Data)
			number of currently loaded page is stored in system variable sp
			used in a page initialize event will auto-send page loaded
			usage: sendme
			<number> 0 sends the value of sp in 0x66 Return Data format
8	sleep		Sleep command. Nextion will ignore touch, and, is not sent to Nextion reader
			sleep 1
			<number> 0 sends the value of sp in 0x66 Return Data format
9	rest		Reset command. Nextion will ignore touch, and, is not sent to Nextion reader
			rest
10	bkcmd		Return command. Nextion sends return data to the serial port.
			<number> 0 sends the value of sp in 0x66 Return Data format
11	touch_j		Recalibrate the Resistive Nextion device's touch screen
			presents a points on screen for user to touch, seven, and then reboots.
			Typically not required as device is factory calibrated.
			sensor can be affected by changes of conditions in environment
			Capacitive Nextion devices can not be calibrated.
			usage: touch_j
			touch_j,1 trigger the recalibration of touch sensor
10	sendxy		Extract character or characters from contents of a flat attribute
			usage: sendxy <attribute>,<start_char>,<end_char>
			<attribute> is text attribute name and will be extracted from
			<start_char> is the character where extracted text will be placed
			<start_char> is optional parameter 0-9 for this command
			<count> is the number of characters to extract
			sendxy t0,0,0,2 Extracts the first two characters from t0.txt into t0.txt
11	vis		Hide or Show component on current page
			Hide will remove component and bring it to the forefront layer
			hide will remove component visually, touch events will be disabled
			use layering with visualize purposes and layering and flashing
			use with caution and minefuls purpose, may lead to difficult debug sessions
			usage: vis <component>
			<component> is either .pkaction id of component on current page,
			vis0 id 0 = page 0 for t0 component exists, and t0.id for t0
			vis0 id 1 = page 1 for t0 component exists, and t0.id for t0
			vis t0,0,0 Hide component with .pkaction id 0
			vis t0,1 Show component with .pkaction id1
			vis t0,2 Toggle component with .pkaction id2
			vis t0,3 Show component with .pkaction id3
12	tsleep		Extract character or characters from contents of a flat attribute
			usage: tsleep <component_name>,<value_to_sleep>
			<component_name> is the component name to be sleeped
			<value_to_sleep> is the amount of time to sleep
			tsleep t0,1 Sleep the component t0 for 1 second
			tsleep t0,0 Wake the component t0
13	tsw		Enable or disable touch events for component on current page
			by default, all components are enabled unless disabled by tsw
			use with caution and minefuls purpose, may lead to difficult debug sessions
			usage: tsw <component_name>
			<component_name> is either .pkaction id of component on current page,
			vis0 id 0 = page 0 for t0 if component exists, and t0.id for t0
			vis0 id 1 = page 1 for t0 if component exists, and t0.id for t0
			tsw t0,0,0 Disable Touch Press/Release events for component t0
			tsw t0,1,1 Enable Touch Press/Release events for component t0
			tsw t0,1,0 Disable Touch Press/Release events for component t0 id 1
			tsw t0,2,2 Toggle Touch Press/Release events for component t0 id 2
			tsw t0,0,2 Disable all Touch Press/Release events on current page
14	com_stop		Stop Serial Component Data Transmission
			Serial will continue to receive and store in buffer
			execution of instructions from Serial will resume with com_star (see **3.0**)
			using com_stop and com_star may cause unintended consequences
			Refer to device datasheet for buffer size and command queue size
			usage: com_stop
			com_stop,1 stops execution of instructions from Serial
15	com_star		Resume execution of instructions received from Serial
			used to resume an execution stop caused by com_stop (see **3.0**)
			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,1 resume execution of instructions from Serial
16	cardset		Set the Nextion Generator Range for use with tsort (see **5.N**)
			range and period until changed or Nextion reboot
			set range to desired range before using tsort
			<range_name> is either a number or a keyword, <number>, custom range is user defined.
			usage: cardset <range_name>
			<range_name>,<number_of_elements_to_sort>
			cardset 0,100 Sets the Nextion Generator Range for use with tsort (see **5.N**)
			cardset 100 Start current random generator range from 1 to 100
			cardset 0,0255 Start current random generator range from 0 to 255
17	cirs_j		Clear the commands/data queued in command buffer without execution
			usage: cirs_j
			cirs_j,1 Clears the command buffer without execution
18	prints		Send raw formatted data over serial (0x70 Return Data)
			prints data not raw Nextion Return Data, use: mset Nextion MCU side
			Use of data may be limited by serial buffer (all data < 1024)
			numeric value sent is 4 bytes (2 16 little endian order)
			value = byte(<type>)*256+byte(<type>)*65536+byte(<type>)*16777216
			text (strlen) sent is strlen +1(0) bytes per characters, without null byte.
			usage: prints <data>,<length>
			<data> is either component attribute, constant or expression
			<length> is optional, if omitted, the length of the data is used instead.
			prints t0,0,2 If return 0 type data then 2 bytes of text data will be sent
			prints t0,0,4 If return 0 type data, 4 bytes per char of text will be sent
			prints t0,0,2 If return 0 type, 2 bytes per char of text will be sent
			prints 0,0,2 If return 0 type, 2 bytes per char of text will be sent
			prints 0,0,4 If return 0 type, 4 bytes per char of text will be sent
			prints 0,0,0 If return 0 type, 0 bytes per char of text will be sent
			prints "Hello",0,4 Sends "Hello" in 4 bytes
			prints "Hello",0,0 Sends "Hello" in 0 bytes
			prints t0.txt,0,2 Sends the contents of t0.txt in 2 bytes
			prints t0.txt,0,0 Sends the contents of t0.txt in 0 bytes
			prints t0.n,0,2 Sends the contents of t0.n in 2 bytes
			prints t0.n,0,0 Sends the contents of t0.n in 0 bytes
			prints 0,0,2 Sends the contents of the current page in 2 bytes
			prints 0,0,0 Sends the contents of the current page in 0 bytes
19	printh		Send raw formatted data over serial (0x70 Return Data)
			printh data not raw Nextion Return Data, use: mset Nextion MCU side
			Use of data may be limited by serial buffer (all data < 1024)
			numeric value sent is 1 byte (0-127 little endian order)
			value = byte(<type>)*128+byte(<type>)*16384+byte(<type>)*2097152
			text (strlen) sent is strlen +1(0) bytes per characters, without null byte.
			usage: printh <data>,<length>
			<data> is either component attribute, constant or expression
			<length> is optional, if omitted, the length of the data is used instead.
			printh t0,0,2 If return 0 type data then 2 bytes of text data will be sent
			printh t0,0,4 If return 0 type data, 4 bytes per char of text data will be sent
			printh t0,0,2 If return 0 type, 2 bytes per char of text data will be sent
			printh 0,0,2 If return 0 type, 2 bytes per char of text data will be sent
			printh 0,0,4 If return 0 type, 4 bytes per char of text data will be sent
			printh 0,0,0 If return 0 type, 0 bytes per char of text data will be sent
			printh "Hello",0,4 Sends "Hello" in 4 bytes
			printh "Hello",0,0 Sends "Hello" in 0 bytes
			printh t0.txt,0,2 Sends the contents of t0.txt in 2 bytes
			printh t0.txt,0,0 Sends the contents of t0.txt in 0 bytes
			printh t0.n,0,2 Sends the contents of t0.n in 2 bytes
			printh t0.n,0,0 Sends the contents of t0.n in 0 bytes
			printh 0,0,2 Sends the contents of the current page in 2 bytes
			printh 0,0,0 Sends the contents of the current page in 0 bytes
20	tsort		Sort command. Nextion will ignore touch, and, is not sent to Nextion reader
			sort 0
			<number_of_elements_to_sort_from_range_set_by_cardset_command_(see_**3.0**)>
			sort 100 Sort 100 elements from range set by cardset
			sort 255 Sort 255 elements from range set by cardset
			sort 0 Sort 0 elements from range set by cardset
			sort 1 Sort 1 element from range set by cardset
			sort 2 Sort 2 elements from range set by cardset
			sort 3 Sort 3 elements from range set by cardset
			sort 4 Sort 4 elements from range set by cardset
			sort 5 Sort 5 elements from range set by cardset
			sort 6 Sort 6 elements from range set by cardset
			sort 7 Sort 7 elements from range set by cardset
			sort 8 Sort 8 elements from range set by cardset
			sort 9 Sort 9 elements from range set by cardset
			sort 10 Sort 10 elements from range set by cardset
			sort 11 Sort 11 elements from range set by cardset
			sort 12 Sort 12 elements from range set by cardset
			sort 13 Sort 13 elements from range set by cardset
			sort 14 Sort 14 elements from range set by cardset
			sort 15 Sort 15 elements from range set by cardset
			sort 16 Sort 16 elements from range set by cardset
			sort 17 Sort 17 elements from range set by cardset
			sort 18 Sort 18 elements from range set by cardset
			sort 19 Sort 19 elements from range set by cardset
			sort 20 Sort 20 elements from range set by cardset
			sort 21 Sort 21 elements from range set by cardset
			sort 22 Sort 22 elements from range set by cardset
			sort 23 Sort 23 elements from range set by cardset
			sort 24 Sort 24 elements from range set by cardset
			sort 25 Sort 25 elements from range set by cardset
			sort 26 Sort 26 elements from range set by cardset
			sort 27 Sort 27 elements from range set by cardset
			sort 28 Sort 28 elements from range set by cardset
			sort 29 Sort 29 elements from range set by cardset
			sort 30 Sort 30 elements from range set by cardset
			sort 31 Sort 31 elements from range set by cardset
			sort 32 Sort 32 elements from range set by cardset
			sort 33 Sort 33 elements from range set by cardset
			sort 34 Sort 34 elements from range set by cardset
			sort 35 Sort 35 elements from range set by cardset
			sort 36 Sort 36 elements from range set by cardset
			sort 37 Sort 37 elements from range set by cardset
			sort 38 Sort 38 elements from range set by cardset
			sort 39 Sort 39 elements from range set by cardset
			sort 40 Sort 40 elements from range set by cardset
			sort 41 Sort 41 elements from range set by cardset
			sort 42 Sort 42 elements from range set by cardset
			sort 43 Sort 43 elements from range set by cardset
			sort 44 Sort 44 elements from range set by cardset
			sort 45 Sort 45 elements from range set by cardset
			sort 46 Sort 46 elements from range set by cardset
			sort 47 Sort 47 elements from range set by cardset
			sort 48 Sort 48 elements from range set by cardset
			sort 49 Sort 49 elements from range set by cardset
			sort 50 Sort 50 elements from range set by cardset
			sort 51 Sort 51 elements from range set by cardset
			sort 52 Sort 52 elements from range set by cardset
			sort 53 Sort 53 elements from range set by cardset
			sort 54 Sort 54 elements from range set by cardset
			sort 55 Sort 55 elements from range set by cardset
			sort 56 Sort 56 elements from range set by cardset
			sort 57 Sort 57 elements from range set by cardset
			sort 58 Sort 58 elements from range set by cardset
			sort 59 Sort 59 elements from range set by cardset
			sort 60 Sort 60 elements from range set by cardset
			sort 61 Sort 61 elements from range set by cardset
			sort 62 Sort 62 elements from range set by cardset
			sort 63 Sort 63 elements from range set by cardset
			sort 64 Sort 64 elements from range set by cardset
			sort 65 Sort 65 elements from range set by cardset
			sort 66 Sort 66 elements from range set by cardset
			sort 67 Sort 67 elements from range set by cardset
			sort 68 Sort 68 elements from range set by cardset
			sort 69 Sort 69 elements from range set by cardset
			sort 70 Sort 70 elements from range set by cardset
			sort 71 Sort 71 elements from range set by cardset
			sort 72 Sort 72 elements from range set by cardset
			sort 73 Sort 73 elements from range set by cardset
			sort 74 Sort 74 elements from range set by cardset
			sort 75 Sort 75 elements from range set by cardset
			sort 76 Sort 76 elements from range set by cardset
			sort 77 Sort 77 elements from range set by cardset
			sort 78 Sort 78 elements from range set by cardset
			sort 79 Sort 79 elements from range set by cardset
			sort 80 Sort 80 elements from range set by cardset
			sort 81 Sort 81 elements from range set by cardset
			sort 82 Sort 82 elements from range set by cardset
			sort 83 Sort 83 elements from range set by cardset
			sort 84 Sort 84 elements from range set by cardset
			sort 85 Sort 85 elements from range set by cardset
			sort 86 Sort 86 elements from range set by cardset
			sort 87 Sort 87 elements from range set by cardset
			sort 88 Sort 88 elements from range set by cardset
			sort 89 Sort 89 elements from range set by cardset
			sort 90 Sort 90 elements from range set by cardset
			sort 91 Sort 91 elements from range set by cardset
			sort 92 Sort 92 elements from range set by cardset
			sort 93 Sort 93 elements from range set by cardset
			sort 94 Sort 94 elements from range set by cardset
			sort 95 Sort 95 elements from range set by cardset
			sort 96 Sort 96 elements from range set by cardset
			sort 97 Sort 97 elements from range set by cardset
			sort 98 Sort 98 elements from range set by cardset
			sort 99 Sort 99 elements from range set by cardset
			sort 100 Sort 100 elements from range set by cardset
			sort 255 Sort 255 elements from range set by cardset

17a	prtd	PRINT	prints [Expr1,] [,Expr2] [,Expr3] to little endian order prints "Hello" if stream 1 bytes for text "Hello" are free prints 5,5,12 if stream 5 bytes for value 555 (0x1B 0x02) **Deprecated**: Send text formatted data over Serial to MCU - print,[prtd] does not use Nextion Return Data, user must handle MCU side - Use of data may be limited by serial buffer (256 bytes - 1024 bytes) - numeric value sent is 4 bytes (32 bit) little endian order - value = 1000 * byte2 + 256 * byte3 + byte4 * 65536 - text content sent is sent 1 MCU byte per character, without null byte - usage prtd, value= =text is either component attribute, variable or literal =text can be a mix of literal, variable and attribute in any order =text literal can be in single, double or back quotes, with no escape sequences print "0.0" // stream 1 bytes for value 0 (0x00 little endian) print "255" // stream 1 bytes for value 255 (0xFF little endian) print 123.12 // stream 4 bytes for value 12 (0x0B 0x00 0x00 0x00)
18	prints	PRINTS	**Send raw data** or multiple raw bytes over Serial to MCU - prints is one of the few commands that preserves user space char 0x20 when more than one byte is being sent a space separate each byte - Data is represented by 1 of 3 ASCII char of hexadecimal value per byte - qty may be limited by serial buffer (all data + 1024) - prints,[prtd] does not use Nextion Return Data, user must handle MCU side - usage prints =hexadecimal,[=hexadecimal],... =hexadecimal = hexadecimal value of each nibble, 0-9, A-F =space= is a space char 0x20, used to separate each =hexadecimal prints 66 C7 and single byte value 255 (0xFF) =FF=0A=FA // sent two bytes value 10,250 (0xA 0xFA) prints 123 // stream 1 byte for value 123 (0x7B) prints 123 255 // stream 3 bytes for value 123 0xFF 0x00 (0x00) - Send raw data or multiple raw bytes over Serial to MCU - prints is one of the few commands that preserves user space char 0x20 when more than one byte is being sent a space separate each byte - Data is represented by 1 of 3 ASCII char of hexadecimal value per byte - qty may be limited by serial buffer (all data + 1024) - prints,[prtd] does not use Nextion Return Data, user must handle MCU side - usage prints =hexadecimal,[=hexadecimal],... =hexadecimal = hexadecimal value of each nibble, 0-9, A-F =space= is a space char 0x20, used to separate each =hexadecimal prints 66 C7 and single byte value 255 (0xFF) =FF=0A=FA // sent two bytes value 10,250 (0xA 0xFA) prints 123 // stream 1 byte for value 123 (0x7B) prints 123 255 // stream 3 bytes for value 123 0xFF 0x00 (0x00)
19	add	ADD	**Add** two bytes or multiple bytes together - usage add =operand=,=operand=,=dest= =operand= is one of the operands =dest= is the destination of the operation =operand= is the channel the data will be added to =dest= is the channel the data will be added to add 123,20 // add value 20 to Channel 0 Waveform with id 123 add 123,0,64 // add 64 bytes to Channel 0 Waveform with id 123
20	cdp	CDP	**Clear** waveform channel data - usage cle =waveform,=channel =waveform= is the id of the waveform component =channel= is the channel to clear cle 123,0 // clear channel 0 waveform id 123 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 // all channel 255 (all channels) cle 12,2 // clear channel 2 data from waveform with id 12 cle 12,255 // clear all channels from waveform with id 12
21	rwd	RWD	**Rewind** the Nextion Device - usage rwd - Rewinds the execution reset of Nextion device - reboot
22	dscmnd	DSCMND	**Send** a Nextion command to a Nextion which port number is specified - useful nextion extension code block for visual refresh (see **3.26** and **3.27**) - usage dscmnd=, =dscmnd= // stream refresh and serial to receive during code block
23	strlen	STRLEN	**Compute** the length of string or text literal - usage strlen =str,=len= =str= must be a string attribute ie =b1.txt, =a0.txt =len= must be numeric ie =n1.val, =a0.val strlen =b1.txt,=n1 // assigns =n1.val with length of =b1.txt content
24a	bitwn	BITWN	**Compute** number of bytes wrong in =str= uses and pad values in =text= - usage bitwn =str=,=len= =str= must be a string attribute ie =b1.txt, =a0.txt =len= must be numeric ie =n1.val, =a0.val bitwn =b1.txt,=n1 // assigns =n1.val with number of bytes =b1.txt encoded
25	if	IF, ELSE, ENDIF	Conditionally execute code block if Nextion condition is true - execute commands within block 1-7 (Conditions) is true - nested conditions using 0 is not allowed, nested 1(1a-c)2-(1)3 - block opening keyword 1 must be on line by itself - no space between block close keyword and first code statement (else /endif) - nested conditions are supported up to 7 levels - usage conditions =expr=,=,=,=,=,=,=,= - conditions can be joined with **&&** or **||** with no spaces used - nested "if" and "else if" suspended - Nextion Conditions are limited to 2 condition blocks[1]else block[2] - (conditions) is a simple non-complex Nextion comparison evaluating left to right - valid (1)(a)=(1)=(2)=(3)=(4)=(5)=(6)=(7)=(8)=(9)=(0)=(a)=(b)=(c) (1)=(0)=(a)=(b)=(c) if =expr=,=,=,=,=,=,=,

		<size_n_bytes_of_the_defined_comp_area>
		<n> is the length of the defined comp area
		<series> is the coordinate of upper left corner of defined comp area
		<n1> is the coordinate of upper left corner of defined comp area
		<p1>is the number of the Resource Picture in the HMI design
		ADR,76,0,0,0

			Upgrade	Returned when power on detects inserted microSD and begins Upgrade by microSD process
31	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)
32	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)

CONTACT INFO

Address: 8F, Building No.9, Banri wowi, Tongsha Road No.32 , Nanshan Dist., Shenzhen, GD, China

Postal Code: 518000 China

Telephone: +86-755-27955416

© Copyright 2011 - 2021 ITEAD STUDIO All Right Reserved