ICE Compiler Commands List v2.0

All the custom tokens and C functions are available by pressing [TRACE] in the program editor. Press the arrow keys to select one or switch menu's.

System

Command	Description
`getKey`	Returns the key being pressed. See *keycodes.png* for the keycodes.
`getKey(KEY)`	Returns 1 if the key `KEY` is pressed, 0 otherwise. This routine is much faster than `getKey`, and handles repeated keypresses as well. See *keycodes.png* for the key codes.
`rand`	Returns a random number between 0 and 16777215.
`Asm(HEX)`	Assembly code written in hexadecimal is inserted into the program.
`AsmComp(PROGRAM)`	Compiles the BASIC program `PROGRAM`. When it's done, ICE continues with compiling the previous program.
`Pause`	Pauses the program until the user presses `ENTER`.
`Pause EXP`	Pauses the program for `EXP` milliseconds.
`prgmPROGRAM`	Executes the BASIC program `PROGRAM`. Any error will be ignored and it will return normally.
`i`	If the imaginary `i` (not the lowercase i) is the first token of a line, this line will be ignored during compiling.

Math

Command	Description
VAR	Returns the value of VAR. Multiple-letter variables are allowed up to 10 letters. Valid letters are A-θ.
°VAR	Returns the address of VAR.
CONST	Returns CONST.
EHEX	Returns the hexadecimal number HEX as an integer. The prefix is the scientific E.
πBIN	Returns the binary number BIN as an integer.
-EXP	Returns the negative of EXP. This is the negative sign, not the minus sign!
EXP1+EXP2	Adds EXP2 to EXP1.
EXP1-EXP2	Substracts EXP2 from EXP1.
EXP1*EXP2	Multiplies EXP1 with EXP2.
EXP1/EXP2	Divides EXP1 by EXP2.
EXP1=EXP2	Returns 1 if EXP1=EXP2, 0 otherwise.
EXP1≠EXP2	Returns 1 if EXP1≠EXP2, 0 otherwise.
EXP1>EXP2	Returns 1 if EXP1>EXP2, 0 otherwise.
EXP1≥EXP2	Returns 1 if EXP1≥EXP2, 0 otherwise.
EXP1≤EXP2	Returns 1 if EXP1≤EXP2, 0 otherwise.
EXP1<EXP2	Returns 1 if EXP1<EXP2, 0 otherwise.
EXP1 **and** EXP2	Returns 1 if EXP1 and EXP2 are both not equal to 0, 0 otherwise.

EXP1 **or** EXP2	Returns 1 if EXP1 or EXP2 is not equal to 0, 0 otherwise.
EXP1 **xor** EXP2	Returns 1 if one of EXP1 and EXP2 is not equal to 0, 0 otherwise.
EXP1·EXP2	Returns the bitwise and of EXP1 and EXP2. This is a plot style token.
EXP1▫EXP2	Returns the bitwise or of EXP1 and EXP2. This is a plot style token.
EXP1▫EXP2	Returns the bitwise xor of EXP1 and EXP2. This is a plot style token.
EXP→VAR	Stores EXP into VAR.
not(EXP**)**	Returns the negation of EXP, so if EXP is nonzero, it returns 0, and 1 otherwise.
remainder(EXP1,EXP2**)**	Returns the remainder of EXP1/EXP2.
min(EXP1,EXP2**)**	Returns the minimum of EXP1 and EXP2.
max(EXP1,EXP2**)**	Returns the maximum of EXP1 and EXP2.
mean(EXP1,EXP2**)**	Returns the mean of EXP1 and EXP2.
sqrt(EXP**)**	Returns the square root of EXP.
sin(EXP**)**	Returns the sine root of EXP. One period is `[0,255]` and it returns a value in `[-255,255]`.
cos(EXP**)**	Returns the cosine root of EXP. One period is `[0,255]` and it returns a value in `[-255,255]`.

Controls

Command	Description
If EXP:code:**End**	If EXP is true, code will be executed.
If EXP:code1:**Else**:code2:**End**	If EXP is true, code1 will be executed, code2 otherwise

`Repeat EXP:code:End`	Repeats executing `code` until `EXP` is true.
`While EXP:code:End`	Executes `code` until `EXP` is true. `EXP` is checked first.
`For(VAR,EXP1,EXP2):code:End`	`VAR` is initialized with `EXP1`. If `VAR` is greater than `EXP2`, the loop ends. Otherwise, `code` is executed and `VAR` is incremented.
`For(VAR,EXP1,EXP2,EXP3):code:End`	`VAR` is initialized with `EXP1`. If `VAR` is greater than `EXP2`, the loop ends. Otherwise, `code` is executed and `EXP3` is added to `VAR`. If `EXP3` is a negative constant, it will loop until `VAR` is smaller than `EXP1`.

Labels

Command	Description
	Labels are limited to 10 characters.
`Lbl` LABEL	Creates a label at the current position.
`Goto` LABEL	Jumps to a label.
`Call` LABEL	Calls a label. This label should have a `Return`, otherwise the program will very likely crash.
`Return`	Returns. If you called a label before, and `Return` is in that label, the program will jump back to the main program. Otherwise, you program will end.
`ReturnIf EXP`	Returns if `EXP` is true. If you called a label before, and `ReturnIf` is in that label, the program will jump back to the main program. Otherwise, you program will end.

Graphics

Command	Name	Description
`det(0)`	Begin	Sets up the graphics canvas (8bpp, default palette).
`det(1)`	End	Closes the graphics library and sets up for the TI-OS.
`det(2,COLOR)`	SetColor	Sets the global color index for all routines.
`det(3)`	SetDefaultPalette	Sets up the default palette where H=L (xLIBC palette).
`det(4,"DATA",SIZE,OFFSET)`	SetPalette	Sets entries in the palette. Each entry is 2 bytes, so `SIZE` should be the amount of entries you want to set times 2.
`det(5,COLOR)`	FillScreen	Fills the screen with the specified color index.
`det(6,X,Y)`	SetPixel	Sets the color pixel to the global color index.
`det(7,X,Y)`	GetPixel	Gets a pixel's color index.
`det(8)`	GetDraw	Gets the current draw location. 0 = screen, 1 = buffer.
`det(9,BUFFER)`	SetDraw	Forces drawing routines to operate on the offscreen buffer or to operate on the visible screen. 0 = draw at screen, 1 = draw at buffer.
`det(10)`	SwapDraw	Safely swap the vram buffer pointers for double buffered output.

`det(11,BUFFER)`	Blit	Copies the buffer image to the screen and vice versa. 0 = copy screen to buffer, 1 = copy buffer to screen.
`det(12,BUFFER,Y,LINES)`	BlitLines	Copies `LINES` lines starting at position `Y` from the buffer to the screen or vice versa. 0 = copy screen to buffer, 1 = copy buffer to screen.
`det(13,BUFFER,X,Y,WIDTH,HEIGHT)`	BlitArea	Copies a specific rectangle starting at (`X`,`Y`) and dimensions (`WIDTH`,`HEIGHT`) from the buffer to the screen or vice versa. 0 = copy screen to buffer, 1 = copy buffer to screen.
`det(14,CHAR)`	PrintChar	Places a character at the current cursor position. (Should be a number, you can find them here. Default coordinates are (0,0).
`det(15,EXP,CHARS)`	PrintInt	Places signed `EXP` at the current cursor position with `CHARS` characters. Default coordinates are (0,0).
`det(16,EXP,CHARS)`	PrintUInt	Places unsigned `EXP` at the current cursor position with `CHARS` characters. Default coordinates are (0,0).
`det(17,"STRING")`	PrintString	Places a string at the current cursor position. Default coordinates are (0,0).
`det(18,"STRING",X,Y)`	PrintStringXY	Places a string at the given coordinates.
`det(19,X,Y)`	SetTextXY	Sets the coordinates for text routines.

`det(20,COLOR)`	SetTextBGColor	Sets the background text color for text routines. Default color is 255.
`det(21,COLOR)`	SetTextFGColor	Sets the foreground text color for text routines. Default color is 0.
`det(22,COLOR)`	SetTextTransparentColor	Sets the transparency text color for text routines. Default color is 255.
`det(25,SPACE)`	SetMonoSpaceFont	Sets the font to be monospace.
`det(26,"STRING")`	GetStringWidth	Gets the pixel width of `"STRING"`.
`det(27,"CHAR")`	GetCharWidth	Gets the pixel width of the char `"CHAR"`.
`det(28)`	GetTextX	Returns the current text cursor X position.
`det(29)`	GetTextY	Returns the current text cursor Y position.
`det(30,X1,Y1,X2,Y2)`	Line	Draws an arbitrarily clipped line.
`det(31,X,Y,LENGTH)`	HorizLine	Draws an clipped horizontal line with the global color index.
`det(32,X,Y,LENGTH)`	VertLine	Draws an clipped vertical line with the global color index.
`det(33,X,Y,RADIUS)`	Circle	Draws a clipped circle outline.
`det(34,X,Y,RADIUS)`	FillCircle	Draws an clipped filled circle.
`det(35,X,Y,WIDTH,HEIGHT)`	Rectangle	Draws an clipped rectangle outline with the global color index.
`det(36,X,Y,WIDTH,HEIGHT)`	FillRectangle	Draws an clipped rectangle with the global color index.
`det(37,X1,Y1,X2,Y2)`	Line_NoClip	Draws an arbitrarily unclipped line.
`det(38,X,Y,LENGTH)`	HorizLine_NoClip	Draws an unclipped horizontal line with

det(38,X,Y,LENGTH)	HorzLine_NoClip	the global color index.
det(39,X,Y,LENGTH)	VertLine_NoClip	Draws an unclipped vertical line with the global color index.
det(40,X,Y,RADIUS)	FillCircle_NoClip	Draws an unclipped filled circle.
det(41,X,Y,WIDTH,HEIGHT)	Rectangle_NoClip	Draws an unclipped rectangle outline with the global color index.
det(42,X,Y,WIDTH,HEIGHT)	FillRectangle_NoClip	Draws an unclipped rectangle with the global color index.
det(43,XMIN,YMIN,XMAX,YMAX)	SetClipRegion	Sets the clipping for clipped routines.
det(44,CLIP_REGION)	GetClipRegion	CLIP_REGION is a pointer to the region bounds. Returns 0 if offscreen, 1 if onscreen.
det(45,PIXELS)	ShiftDown	Shifts whatever is in the clip down by some pixels.
det(46,PIXELS)	ShiftUp	Shifts whatever is in the clip up by some pixels.
det(47,PIXELS)	ShiftLeft	Shifts whatever is in the clip left by some pixels.
det(48,PIXELS)	ShiftRight	Shifts whatever is in the clip right by some pixels.
det(59,INDEX,X,Y)	Sprite_NoClip	Places an sprite on the screen as fast as possible.
det(60,INDEX,X,Y)	TransparentSprite_NoClip	Draws a transparent sprite to the current buffer. The transparent color is 0.
det(62,INDEX,X,Y,WSCALE,HSCALE)	ScaledSprite_NoClip	Draws a scaled sprite to the screen.

`det(63,INDEX,X,Y,WSCALE,HSCALE)`	ScaledTransparentSprite_NoClip	Draws a scaled sprite to the screen with transparency. The transparent color is 0.
`det(71,X1,Y1,X2,Y2,X3,Y3)`	FillTriangle	Draws a filled triangle with clipping.
`det(72,X1,Y1,X2,Y2,X3,Y3)`	FillTriangle_NoClip	Draws a filled triangle without clipping.
`det(74,XSCALE,YSCALE)`	SetTextScale	Changes the amount of text scaling (note that height and width are independent).
`det(75,COLOR)`	SetTransparentColor	Sets the global transparent color index for all routines.
`det(76)`	ZeroScreen	Fills the current screen with a bunch of zeros. Same as det(5,0) but faster.
`det(77,CONFIG)`	SetTextConfig	Configures text depending on the arguments. 1 = clipping, 2 = no clipping.
`det(81,HEIGHT)`	SetFontHeight	Sets the height of the font in pixels.
`det(83,X,Y,COLOR)`	FloodFill	Fills an area with a color.

Memory

Command	Description
`DefineSprite(WIDTH,HEIGHT)`	Allocates `WIDTH*HEIGHT+2` bytes for an empty sprite. This is recommend for duplicating, rotating or enlarging sprites. Returns a pointer to the allocated memory.
`DefineSprite(WIDTH,HEIGHT,"DATA")`	Defines a sprite with data `DATA`, width `WIDTH` and height `HEIGHT`. Returns a pointer to the start of the sprite.
`Data(SIZE,CONST1,CONST2...)`	Puts all the constants in the program data with size `SIZE`. Returns a pointer to the data.

Alloc(SIZE**)**	Allocates SIZE bytes in safe RAM. An error will be displayed if you allocated too much memory. Returns a pointer to the memory.
Copy(DEST,SRC,SIZE**)**	Copies SIZE bytes from SRC to DEST. **rand**, **getKey** and **getKey(**KEY**)** are now allowed as a standalone argument.
Copy(...,DEST,SRC,SIZE**)**	Copies SIZE bytes from SRC to DEST backwards. The first argument can be anything you like. **rand**, **getKey** and **getKey(**KEY**)** are now allowed as a standalone argument.
L_1...L	Returns the pointer to one of the OS lists as an integer.
LIST(EXP)	Same as {LIST+EXP} but easier to remember.
*{EXP}	Returns the single byte EXP points to.
**{EXP}	Returns the 2-byte value EXP points to.
{EXP}	Returns the word EXP points to. ***{EXP} is valid syntax as well.
EXP1→*{EXP2}	The single byte of EXP1 is stored where EXP2 points to.
EXP1→**{EXP2}	The 2-byte value EXP1 is stored where EXP2 points to.
EXP1→{EXP2}	The word EXP1 is stored where EXP2 points to.

Strings

Command	Description
"STRING"	Adds a string to the program data and returns a pointer to it.
OS_STRING	Returns a pointer to OS_STRING.
"STRING1"+"STRING2"	Concatenates two strings. OS strings are valid as well. Returns a pointer to the data.
	Stores "STRING1" into one of the 10 available OS strings. Str0

"STRING1"→OS_STRING	Stores "STRING1" into one of the 10 available OS strings, Str0-Str9. Storing one OS string into another one is valid as well.
sub("STRING",OFFSET,SIZE)	Copies SIZE bytes starting at offset OFFSET from STRING to a temporary location and returns a pointer to it. OS strings are valid as well.
length("STRING")	Returns the length of STRING. OS strings are valid as well.

Text

Command	Description
Disp EXP1,EXP2,...	Displays either an expression or a string at the cursor position. *i* puts the cursor at the start of a new line.
Output(ROW,COLUMN)	Puts the cursor at coordinates (ROW,COLUMN).
Output(ROW,COLUMN,EXP1)	Displays either the outcome of an expression or a string at coordinates (ROW,COLUMN).
ClrHome	Clears the homescreen full. It is recommend to call this function before using Disp. The cursor moves to the upper-left corner.
Input VAR	Asks for a user input, and stores the value in VAR.