

# **C64 DEV MACHINE**

Visual IDE for Commodore 64 Development

## **Getting Started Guide - Revision 1.260404**

---

*Some C64 Coding knowledge may be required, but let's dive in!*

---

Foreword:

In 1986 I got my first Commodore 64, it was there in my little cubby hole that I learnt Basic, played Action Biker, Ghost Chaser, Spy Hunter and Bruce Lee to name but a few. Those were some very peaceful times indeed until 1 year later games like Last Ninja, Fist II and Leaderboard started to emerge. Those games truly inspired me when I was young, I wanted to know how they were made but there was very little to know other than occasional Poke codes you'd see in magazines like ZZAP64 or Commodore Format. Those were mostly big assembly data blocks that somehow magically worked, don't ask me how but they did.

A few years later I got an Action Replay MKIV cartridge, it was a magical thing that let you see inside a game by freezing it, looking at the code, sprite data, everything really... I learnt how to change things, replace sprites for my own ones created using some of the PD disk tools I used to order, or some from Commodore Disk User magazine. It was magic to me, but this was around 1992 now and the Amiga was becoming popular. I moved on to using Blitz Basic 2.0 on Amiga and was starting to make games that way, but again, things moved on and then we got a home PC around 1998 where I moved to making 3D graphics.

Now, 40 years later, celebrating 40 years of owning a C64, not only have I been buying up these old machines but also fixing them, cleaning them up, collecting games and now, creating tools so you and I can finally make those games we always wanted to.

Robert Ramsay

© Polytricity Ltd

# 1. What is the C64 Dev Machine?

C64 Dev Machine is a **visual programming tool** for making software on the **Commodore 64**, that beige home computer from 1982 that is still beloved by developers, musicians, and artists worldwide. Instead of writing lines of assembly code by hand, you connect visual blocks called nodes together on a canvas. When you press F5, it compiles your node graph into real machine code and launches it in VICE, a built-in Commodore 64 emulator.

**i Note:** You don't need to know everything about programming on a C64 to get started, I have made several tutorials that will help you, as well as custom macros that take care of the big stuff!

But if you do want to learn more, please prepare your submersion vehicle!...

<http://www.6502.org/tutorials/6502opcodes.html>

<https://c64os.com/post/6502instructions>

<https://www.youtube.com/@board-b-tutorials>

## How it works ... in one sentence

---

You build a chain of nodes, press F5, your program runs on a C64 emulator.!

## 2. The Interface at a Glance

When you open C64 Dev Machine you'll see these main areas:


Key / Action	What it does
<b>Canvas</b>	The large dark area where you build your node graphs.
<b>Opcode table</b>	A toolbar containing instruction buttons in 3 pages.
<b>Macros</b>	A list of available and ever growing macros that make life easy.
<b>Variables</b>	Various variable types you name and they auto allocate to memory.
<b>Memory Bar</b>	A coloured bar at the bottom showing how your C64 memory is being used.
<b>Shortcuts</b>	A list of shortcut buttons and their shortcut keys.
<b>Asset manager</b>	Here you can add new assets to be used in the nodes.

## Viewport Navigation

---

Moving around the canvas is simple:

Key / Action	What it does
Middle/Right mouse button, Alt, or arrow keys	Pan the canvas in any direction.
Mouse wheel, Page up/page down	Zoom in and out.
Press 1,2,3,4,5	Quick-change zoom Level
Double-click on mapping boxes	Zoom to and focus on a mapping box
Double-Right-click on a block	Delete the node or variable or empty ORG block
Home key	Snap the view back to INIT
Spacebar	Show list of mapping box label and pick to zoom to them

 **Tip:** If you get lost, press Home to jump back to the beginning of your project.

## 3. Nodes - The Building Blocks

Everything in the C64 Dev Machine is a **node**. A node is a self-contained block that does one thing, a single instruction to the exact byte or to play a SID track, move a sprite, wait for the screen to draw a frame, determine your joystick input and more.

Nodes connect together in a top-down chain or spine. You can spawn ORG blocks that continue the chain but allow you to code to the right, or set the ORG blocks start ADDRESS by disabling the proxy checkbox and you code anywhere in memory. The program runs through each spine from left to right when your C64 software starts. ORG blocks can overwrite the INIT block if you disable its proxy and set its address to \$080E. Be careful where you place things in memory as it's too easy for things to overlap. There is a feature that will flash overlapping nodes (culprits) red if they are involved in some kind of overlapping of their memory allocation or size.

### Creating Nodes

---

There are two ways to add nodes to the canvas:


Key / Action	What it does
Drag them over from their respective button panels.	Click and drag any opcode, macro or variable into the work area or direct to a spine and it will snap into place.
Press their first letter combined with a second letter while hovering over them.	For example, pressing J will spawn a JMP node at mouse position, pressing J on it again will convert it to a JSR. Similarly RTS by pressing R and then pressing R while hovering over it will change it to an RTI.
Quick shortcuts	L = LDA_IMM , S = STA_ABS , Alt+V for a VWAIT, I (i) for IF Node
Vars by clicking	Click on one of the VAR types to create it, name it and it will latch to a VARIABLES Org block.

## Connecting Nodes

- **Left-click and drag** from the panel, let go near the base of the INIT(Green) or ORG(Purple) spines and let go.
- Assets are looked up from a node once added to the assets
- Variables only attach to Variable ORGs (Blue)
- Right click a node to delete it, all other nodes will truncate upwards.

## Moving and Managing Nodes

Key / Action	What it does
<b>Left-click drag</b>	Move a node around the canvas.
<b>Ctrl + left-click drag</b>	Clone (duplicate) a node. A copy snaps to your cursor.
<b>Right-click</b>	Instantly delete the selected node.
<b>Click node part</b>	Edit its property or value.
<b>Ctrl+Drag in space</b>	Select multiple nodes, drag the top-most one to move them, ctrl+drag to clone them or press delete to delete them.
<b>Click in space</b>	To deselect

 **Tip:** Hold CTRL and drag any node to make an instant copy - handy for repeating patterns or you just need another node of the same kind.

## 4. Keyboard Shortcuts and Variable types

These shortcuts work when your mouse is over the canvas. Each key drops a node at your cursor position.

### Structure Nodes

Key / Action	What it does
<b>O</b>	ORG - sets a new memory origin address or continues by proxy.
<b>A</b>	ADDRESS LABEL - creates a named jump target for branches and loops.
<b>J</b>	JMP Node - Common node, press J on it to toggle to JSR
<b>C</b>	COMMENT -adds a text note to the canvas (doesn't compile).
<b>V / ALT+V</b>	Variables Org block - for organizing variables. Alt+V for a VWait Node
<b>B</b>	Create a mapping box - you can later search for these by pressing space.
<b>R</b>	Spawn an RTS node - toggle to an RTI by pressing R over it
<b>I</b>	IF Node for testing values against variables
<b>L / S</b>	Spawn LDA_IMM or STA_ABS (Most common)

## Variable Node Definitions

Key / Action	What it does
<b>SET VAR</b>	SET_VAR - store a value into a named variable.(use ABS or REL for absolute value or relative to itself for POSitive or NEGative to what it was)
<b>GER VAR</b>	GET_VAR - load a variable's value into the Accumulator register.
<b>INC VAR</b>	INC_VAR - Increase a variable by 1 (cheaper than set var with REL POS 1)
<b>DEC VAR</b>	DEC_VAR - Decrease a variable by 1 (cheaper than set var with REL NEG 1)


## Variable Types

Key / Action	What it does
<b>BYTE</b>	A byte ranges from 0-255 (00-FF) (unsigned)
<b>WORD</b>	A word ranges from 0-65535 (0000-FFFF) (unsigned)
<b>BCD</b>	A binary coded decimal is 3 bytes each 0-99 for scores 000000 - 999999
<b>STR</b>	A string holds an array of characters defined inline or by asset reference.
<b>BYTE_S</b>	Ranges from 0-255 (00-FF) (SIGNED : entering -\$01 or -1 will be \$FE)

## IF Node

Key / Action	What it does
<b>== (BEQ)</b>	Test 2 values if they are equal jump to the location defined.
<b>!= (BNE)</b>	Test 2 values if they are NOTE equal jump to the location defined.

The IF NODE has a special feature for reaching further into code called spring boarding. Normally a regular branch (BNE / BEQ) can only reach -128 or +127 away from itself so in that case you can enable springboarding which will do a branch to a jump from within itself. It costs a few more bytes to do this, so enable it if you need it.

 **Tip:** In some cases you don't need to use the IF Node, for example, it's cheaper if you are checking if a value == or != to zero, you don't need the CMP 00 instruction so the whole instruction set becomes cheaper. This is a feature with the ZERO FLAG register.

## 5. Macro Walkthroughs

Macros are **high-level nodes** that generate many lines of machine code automatically. You just fill in a few settings and the macro handles everything else.

Here's how to use the most common ones:

### **MACRO\_SID - Play Music**

---

Plays a SID music file in the background using IRQ-driven playback. A CIA timer fires 50(PAL - assumed)/60(NTSC) times per second, automatically calling the SID player so your main program doesn't have to.

#### **MACRO\_SID : Step by Step**

1. Add the macro near the INIT Node (or an area that will only run once)
2. Click to add a SID ASSET on the Assets shelf, it will load to its own predetermined location, typically \$1000.
3. Assign the SID asset to the node and select the track (Track 0 is the first track)
4. Note - This Macro needs an exit point so creates one just below the node, keep it there.
5. The node will show the title referenced to the Asset.
6. To hear the SID running you need to add a game loop, add a label(A) and a JMP(J)
7. Press F5 to compile and hear your music playing in the emulator.

### **[Watch the Tutorial on the Macro SID](#)**

**i Note:** SID files are the standard music format for the C64. Thousands are freely available at [hvsc.c64.org](http://hvsc.c64.org) the High Voltage SID Collection but not all are prepared to run and may result in a crash. SID tracks can be created using GoatTracker.

## **MACRO\_SPR - Display Sprites**

---


Sprites are small moveable graphics, up to 8 can be displayed per frame, although Raster tricks can allow for more, we will keep it simple. This macro allows you to set-up and display one of them at a chosen x and y pixel position on screen.

### **MACRO\_SPR - Step by Step**

1. Drag in the Node and attach it to the spine, it only needs setting up once, and not in a game-loop.
2. With a sprite asset loaded (use Spred64 which is built-in to make sprites) point to the asset.
3. Choose which sprite to show
4. Sprites will automatically be enabled with an internal OR function that switches them on as you use this node, should you wish to switch them off, use the build-in Variable explained in a different section.

### **[Watch the Tutorial on the Macro SPR](#)**

*Extra learning about sprites: [How does sprite priority work on the Commodore 64?](#)*

 **Tip:** You can have up to 8 sprites on screen at once, just add more MACRO\_SPR nodes, but be sure to set their SLOT. Slot 0 will be drawn over slot 1 and 1 over 2 and so on.

## **MACRO\_TEXT\_SCROLL - Scrolling Text**

Creates a classic hardware-smooth horizontal text scroller — one of the most iconic effects on the C64.

### **MACRO\_TEXT\_SCROLL - Step by Step**

1. Click TXT\_SCRL on the shelf and drag it onto the canvas.
2. Click INLINE or ASSET
3. Type your scroll message into the text field. Limited to 40 chars unless you use a TEXT ASSET added from the Assets panel.
4. Set the scroll speed (1 = slowest, 7 = fastest).
5. Set the row where the scroller appears.
6. If you have a MACRO\_SID on the same spine, the scroll will automatically sync with the
7. Use the NOP DELAY to finetune any bad lines
8. You can enable JSR mode if you need to use other functions in your game loop, it will generate and alias but you make up a new alias name for it and use that as the JSR call

### SCROLLER COMMANDS:

While writing your string, you can use these commands that don't take up any space or show up in your message but will control certain things.

`_col00` = set the text colour 00 to 15

`_spd01` = set the speed 01 is slow 07 is fast

`_wait` = a short pause, use multiple times for a longer pause.

`_trk00` = set or re-trigger a music track (if added), 00 is the first track, 01 the second etc

**i Note:** The text scroller uses the C64's hardware fine-scrolling feature - it runs entirely in the background via an interrupt, so your main program keeps running at the same time however this affects certain things like joystick input and the JSR version is needed.

## **MACRO\_VIC - Setting up the Vic Bank**

Set the display mode and bank.

### **MACRO\_VIC - Step by Step**

1. The Vic bank determines which area the C64 'sees'. Since it can only see 16kb at one time, enough for a bitmap and about 80 sprites.
2. By default The Charset and Map should be setup to use Bank 0
3. In some cases like Bitmap and Sprites, they are preferred to be in Bank 1
4. The VIC bank is not always required to set up Bitmaps, that is automated

## **MACRO\_BMP - Full-Screen Bitmap**

---

Displays a full-screen Koala Painter format bitmap image, perfect for loading screens and title art.

### **MACRO\_BMP - Step by Step**

1. Click the BITMAP macro button on the shelf and drag it onto the canvas.
2. Front he node choose the bitmap KLA or KOA asset you imported (see assets).
3. The memory map bar at the bottom will update to show where the bitmap is stored.
4. You will need a game-loop to hold the bitmap in place.


## **MACRO\_VWAIT - Frame Sync**

---

Waits for the TV/monitor to finish drawing the current frame before your code continues. This prevents flickering and tearing.

### **MACRO\_VWAIT - Step by Step**

1. Press V to drop a VWAIT node, or click VWAIT on the shelf.
2. Place it inside any loop in your spine - after your game logic, before the loop jumps back.
3. Refine the value if required

 **Tip:** Always put a VWAIT inside game loops to keep animations smooth. Without it, sprites can flicker or tear as they will move faster than the screen refresh rate.

## **MACRO\_MAP (Used with Macro\_Scroll (JSR required) )**

---

Renders a tile-based scrolling map on screen - ideal for platform games and top-down levels.

### **MACRO\_MAP - Step by Step**

1. Click the MAP macro on the shelf and drag it onto the canvas.
2. Link your Map asset from the assets
3. Requires the charset node for custom graphics (link the asset there too)
4. Set the starting X and Y scroll positions.
5. If you need scrolling use the Macro\_Scroll

**FULL TUTORIAL COMING SOON!!**

## 6. Core Function keys / Load+Save / Build

Once your nodes are connected, running your program is simply hit F5

Key / Action	What it does
<b>F1</b>	Cleanup and rogue labels and comments
<b>F2</b>	Debug dump outputs a note form disassembly of your compiled code for inspection.
<b>F3</b>	Hide comments (toggle)
<b>F4</b>	Export the .PRG file
<b>F5</b>	Build and Run, auto launches VICE
<b>F9</b>	Reset/Restart application as if launched.
<b>F11</b>	Toggle Fullscreen modes
<b>CTRL + L/S</b>	Standard Load and Save
<b>HOME</b>	Reset Camera view to the INIT node
<b>CTRL + Z/Y</b>	Undo / Redo (when pressing CTRL you can see undo state)
<b>X</b>	Toggle Decimal or Hexadecimal mode
<b>ESCAPE</b>	Exit current window or application exit

### What happens when you press F5

- The compiler walks down each spine from left to right starting with INIT
- Each node contributes its machine code instructions.
- All labels and addresses are resolved automatically.
- A .PRG file is written to disk.
- VICE (the C64 emulator) launches and runs the PRG automatically.


**i Note:** VICE is integrated so no need to launch it, but you can generate the .PRG with F4 to share with friends.

### The Memory Map Bar

At the bottom of the screen is a colour-coded bar showing how the C64's 64KB of memory is being used:

Key / Action	What it does
<b>Cyan region</b>	Your INIT spine : the main program code starting at \$080E.
<b>Green regions</b>	ORG blocks — additional code segments you've placed.
<b>Yellow regions</b>	Asset data — SID music, bitmaps, sprites, charsets.
<b>Purple regions</b>	UV Variables — your named variables stored at \$C000.

Key / Action	What it does
Red bar	Hardware registers — the C64's built-in chips (VIC, SID, CIA).

 **Tip:** Watch the memory map as you add nodes — if two coloured regions overlap, you have a memory conflict and your program may not work correctly.

## 8. Tips and Troubleshooting

### Common issues

Key / Action	What it does
General crashing	JSRs need existing labels, RTS or Game-Loop required, memory overlaps.
Sprite doesn't appear	Make sure you link to a sprite asset.
Music not playing	Make sure the track has data and the music is loaded to \$1000 (preferably)
Nodes are flashing	Two assets are fighting for the same memory.

### General Tips

- Name your LABELs clearly : 'main\_loop' is better than 'label1'.
- Use COMMENT nodes to document what each section of your spine does.
- Save often. There is an autosave feature every 2 minutes.
- The memory map bar is your best debugging tool, check your locations.
- Use the video tutorials provided.

## 9. Quick Glossary

Key / Action	What it does
VICE	A C64 Emulator
Spine	A chain of connected nodes that compiles into a program.
Node	A single block in a spine. Each node generates machine code.
Macro node	A high-level node that generates complex machine code automatically (SID, sprite, scroll, etc).
PRG	The compiled binary file that VICE runs.

# 10. Assets

## The Asset manager

---

Certain nodes require assets such as bitmap, sprite, SID and optionally scrollers etc.

Asset	What it does
<b>BITMAP</b>	Holds a .KLA or .KOA file stored at \$4000 by default
<b>SPRITE_SET</b>	Holds up to 64 sprites, directly editable with SPRED64 built in but separate application, auto saves and reloads any edits.
<b>SID_MUSIC</b>	Make sure the track has data and the music is loaded to \$1000 and has SFX support. Goat-tracker is preferred for making SID files.
<b>SFX_DATA</b>	Instruments (SFX) are parsed automatically from a Goattracker .sng file
<b>CHAR_SET</b>	Character set data obtained from a binary charset file (Char-pad preferred)
<b>MAP_DATA</b>	Map data obtained from binary map file from Char-Pad, can be edited within C64 Dev machine. Map width Max is 160, Max height is 25.
<b>TEXT_DATA</b>	Holds strings and instructions (see text scroller)
<b>BYTE_DATA</b>	Holds comma separated decimal or hex data e.g: 255 , \$FF, 64, \$40 etc

---

*Happy coding on the 64.* 

© Polytricity Ltd