J74 Assistant

Developers Tools

Table of Contents

About the "J74 Assistant Developers Tools" user manual

This user manual describes content, use cases, setup and instructions for use of the "J74 Assistant Developers Tools" package, a toolset for external automation and for AI integration of Ableton Live (version 11 or higher, running with Max for Live license, as in Ableton Live "Suite").

This document is highly technical. This is due to the nature of the system integration involved. It requires the user to perform setup and configuration tasks to get the environment running and to understand to a certain extent data formats and scripts usage. It does not require you to code anything, though.

This user manual is completed by additional documentation, also part of the set. When such additional documentation plays a role, it is explicitly mentioned in this manual. You can of course review the additional documentation independently, but please take this user manual as you main "story line", as this document goes through the various aspects in a logical fashion.

Please read this manual carefully. When it goes to "Installation and Setup" tasks, please (try to) perform such tasks as close as possible to the given instructions. If you need or decide to deviate from these specifications, please be aware that the maker of this toolset might not be able to support your custom setup.

[Helper: if you are not interested in background information you can skip the next section and proceed to the #"Installation and Setup" section]

An introduction to the "J74 Assistant Developers Tools" package

If you ever crossed path with my projects before, you could probably agree that, in what I do, I try to bring something original. Sometimes it comes from an idea predating even the "DAW times" (yes, I am that old), something I finally found a way to give life to through some form of computer trickery, or maybe it is just a somehow new interpretation of a modern technique, for which I have found a personal and new perspective. In essence, I try to bring something new and exciting to music technology (at least, for myself, I try).

This time it is a bit different.

This time is not about being original. There is no semi-unique musical technique at play or some personal re-interpretation of a musical concept. This time is all about "*bridging technologies*": the technology of music production (Ableton Live) and something else technological, something which, although hyped and at times overblown, it is truly game-changing on the long term, in this field like in every other: *Artificial Intelligence* (AI).

Why? And How?

Let's start with the "*Why?*" (or better, the "*Why NOT?*").

A world without augmentation from AI assistance, based on pure, home-grown, human-only creativity and craftsmanship is a romantic image. I can relate to it. I was born in such a world. But it is also an image of the past. Like it or not, as I write these lines, as you read them, more AI's keep coming in our lives, making a bigger impact on our way of living.

What to me is most staggering regarding AI's, it is not the perspective (and fear) of AI's "replacing" humans in our work (I do not believe that to be truly possible, professionally speaking), but rather it is the productivity boost that AI assistance can bring to human endeavors. It gives our minds and our hands an incredible *extension of powers and capability*. I have seen it already in other fields. And I am sure it will come to music production too.

AI integration is happening and it is happening at a speed which is breathtaking. In the specifics of the IT and system integration worlds (areas in which I spend most of my professional time) I see it everyday growing and it grows fast: AI augmentation to human operation and engineering of "computer things" is already reality today. From a pure productivity standpoint, it is simply incredible to see it in action. It makes our IT systems, machines, engineering faster, more capable, self-healing, controllable.

Yes, the use of AI does bring risks with it. That is undeniable. An AI can hallucinate, think it is right when it is not, its memories are rooted in our mistakes, therefore biased by the things we have done in the past and left to AI's to learn from. It has no true morality (as it has no true life). If you leave it unchecked, on certain routes, an AI might end up wiping out all the work you have done, simply because it may find it not to match a declarative "intent" you gave, deciding to start from scratch. Unchecked, an AI can become "raw", even dangerous.

At the end of the day, I believe that we, as humans, are left with just one option when it comes to AI: *to embrace it, control it and shape it to our benefit*. It might be a questionable position for some (why not just deny it, rule it out, forbid it and go back, if such great dangers could come with it?), but in the end, I believe, it is simply an unavoidable path for us all. Therefore (speaking for myself now) I embraced it. And at that point you should see where my position goes: *why not bring AI integration also to my other interest, "music technology"? Why not?*

But then, *how*? How can you "bridge" an AI to an existing music production environment (such as Ableton Live)?

Let's start by looking at Ableton Live, a relatively modern Digital Audio Workstation (DAW). Ableton Live has not been made, in its current form, with native AI integration in mind. At its origin (in the early 2000's) AI was just a distant sci-fi promise. But some aspects of Ableton Live have grown through the years in such a way that, by linking the existing puzzle pieces, a "controlled hack" and, that way, *the use of AI with it*, becomes possible.

Side-note: For certain use cases even the VST (or AU) protocols could be used (and be abused) for AI integration, if the scope is intentionally limited (a VST can only work on MIDI and audio, on the track it is hosted on). For instance, MIDI note creation (but not processing) and aid to audio parameter selection (e.g. in mastering) is possible with a VST (or AU) plug-in. A few commercial platforms embraced this approach. In my opinion that is limited, too limited by the VST (and AU) protocol. Limited in practical uses. Therefore, that is a path I decided not to explore. What I really wanted to explore is some deeper level of integration.

In the specific of Ableton Live, the Live application has evolved, from version 8 onwards (somewhere in the mid 2010's), with a flexible API (an Application Programming Interface) and a companion object model (the LOM, Live Object Model, a model which describes Ableton Live itself from a programmatic point of view). Together LOM & API allow developers (and "power users") to (re)program and control many of its functionality, *from the inside*.

The API & LOM combined have been exposed by Ableton to two different (internal) development platforms, both (power) users and developers can access: *control-surfaces scripting* (supported by a built-in Python interpreter within the Live application, although officially limited to hardware control surfaces) and *Max for Live* (a programming environment, integrated within Live, which can be used to extend its capabilities in the form of new Max for Live plug-in's, devices for both audio and MIDI applications). I have created with and extensively used, both. I know them quite well.

That experience gave me a start for my research: it was definitively thinkable to connect the *outside world* (and therefore an AI Tool) to the Ableton Live API (and LOM), if you could find a way of *passing through* information and, through this, reach the API. That is exactly where I started from: find a way to connect and standardize an external connection so that an external application (like an AI Tool) could do actions, get information, change state within the rules of the Ableton Live LOM API.

But it is not where I stopped.

Let me go through my experience (you can definitively skip the rest of this introduction if you wish, but from a developer perspective, it might be useful for you to read about my own experience and my choices):

- It indeed all started with making a "*communication bridge*", so to let the outside world access the API of Ableton Live. For that I decided to use OSC (Open Sound Control) messages, a form of networking using messages, purposely limited to the local-host (the local computer, so no external networking is involved – from a security point of view a must for me. Communication must stay confined to the computer Ableton Live runs on). It was indeed possible (and has been done before): with OSC messaging, both in control-surfaces and Max for Live (often used for specific tasks, not AI integration) you can reach inside Ableton Live. In my case I decided to use Max for Live for this "OSC Bridge" functionality, as Max for Live, in general,

is better supported and offers the possibility of a GUI. It is also easier to troubleshoot. Plus, I had plans for more functionality. So, it was for me the better choice.

- The irony is that I then turned to Python (on the host, not within Ableton Live) for another piece of the puzzle: have an external software interface to Ableton Live which an AI could use, a Python environment which could "talk" API via OSC, from the outside (inside your computer, but outside Ableton Live) and reach the inside of Ableton Live. Python is ideal for that: it is an interpreted scripting language, available on all modern OS systems (so also readily available on any computer used for music production today). And it is easy to program. In Python I made several (initial) scripts for controlling the "atomic" API operations (call / get / set) through OSC. These operations required both state knowledge and state manipulation, so they had to work bidirectionally (via the OSC bridge).

- What I discovered in this phase is that having scripts to transparently engage with Ableton Live LOM API from the outside (of Ableton Live) was not enough. Yes, with API calls via OSC you could start or stop a MIDI clip from the outside (ooh what fancy!!), but you could not process or program notes into MIDI clips, or warp markers in audio clips or do anything "truly" useful. More complex operations rely on *data structures* and on *data operations.* So, I had to create a "*data bridge*" for that, so that data which an API-call points to (data inside Ableton Live, structured as dictionaries for MIDI notes, warp markers, device state, mixer state, etc.) could be retrieved, packed, unpacked and pushed from the outside.

- That (data structuring, data unpacking, data packing) was what I did next. To do it I decided to adopt a well known (Internet-wide) data format style, called JSON (JavaScript Object Notation), as it is native to both Max for Live and Python and it is used by the API too. In this way data could be exchanged and programmed through OSC (via the Max for Live engine), again using Python with OSC from the outside.

- At this point I had the basic puzzle pieces ready and indeed with some more tweaking (on the AI Tool side) I was able to get a "local-coding-enabled" AI Tool (like Gemini CLI, an AI which can run or program code and has access to the local computer file system, a sort of code co-developer AI assistant) to talk and operate on Ableton Live. That was immediately fun!

Side-note: in all this "bla bla bla" I do, I tend to categorize AI Tools in "two" families, based on how they interface to humans ***and*** on how they operate on the computer the human has access to. There are, of course, many more ways to categorize AI's, but this is an important distinction for my use cases and for developers.

Let me define these two AI Tool categories:

[*"local-coding-enabled" AI Tools*] These are AI Tools with (almost) full powers on "your" computer. They can run or even create code on your computer and can read and write files on it. Examples are Gemini CLI and Claude Code. This kind of tool can do everything directly on your computer, but they require the user (the human, in this case the producer) also to be a "system integrator", have scripting knowledge and carefully guide the AI in coding, in structuring integration and data. It is a powerful type of AI Tool but requires integration skills and care.

[*"Web" AI Tools*] These are the common AI tools most people know and use today. They are AI's "confined in a bottle" (= the web browser). These AI's cannot run or write code on your system: they can suggest code, as plain text, but they cannot write it and most of all cannot run it. They can interact with humans only through the web browser. That's it. While they are limited in reach, these Web AI tools require no system integration skill.

- Another side-note: while making a Python environment for interfacing an AI with the Ableton Live API through OSC, as a remarkable side effect, I ended up with a powerful Python environment (or "toolset"). This toolset proved to be particularly useful *(in itself)* for all kinds of automation purposes. With it (and without any AI) you can automate Ableton Live actions, actions which otherwise would take a long time to do manually, just by pointing and clicking. Once I realized the power of those tools, I took some extra time to make the Python toolset as flexible and capable as I could, not only in doing "atomic actions" (LOM API calls via OSC) but also in handling *data.* The toolset covers now collection, interpretation, processing and creation of MIDI notes in MIDI clips, warp markers in audio clips, device and mixer parameter snapshots and more. Coupled with the OSC Bridge, it completely covers an interesting use case, what is later

addressed as [use-case 1]: “use of the Python toolset for automation and scripting of Ableton Live actions”. An integral and valuable use-case for the package.

- Returning to AI integration, all the above was tailored to a “local-coding-enabled” AI Tool and with some effort in work-flow automation, standardization and prompt engineering for a specific AI Tool (Gemini CLI) the interaction with the AI became fluent. [use-case 2]: “use of a local-coding-enabled AI Tool such as Gemini CLI as an assistant” became reality. The package was now complete: without additional code you can now ask an AI assistant (Gemini CLI), with common language, to co-work on a musical production task.

- Something else I discovered along the path: having a “data bridge” in place, got me to another possibility. Why, as an alternative to AI work-flow automation leaning on a full scripting environment in Python and OSC communication, not “just exchange raw data” with a common Web AI Tool? Could you do something truly useful (e.g. MIDI creation and even MIDI manipulation) just with a Web AI Tool and “copy & paste”?

- With that in mind I worked on creating data tools (in Max for Live) for pure JSON data “copy & paste” actions. That was relatively easy (as Max for Live has its own scripting capabilities, in JavaScript). Once that worked it was just a question of working on some prompt engineering for a Web AI Tool, making it aware of the JSON data format. Surprisingly it worked quite well! So, NO system integration, NO OSC, NO programming, No Python. Just simple “copy & paste” actions performed by the user (at the end JSON is just “text”) and structured prompts on a Web AI tool. That’s it. That work-flow has become a separate package, called the “J74 MIDI Assistant”, which is essentially a Max for Live plug-in for COPY & PASTE of MIDI clip notes in JSON format plus a guideline to prompt creation for a Web AI Tool. A separate package, which a user with no system integration experience could use with any common Web AI tool in a browser to generate ***and*** to manipulate MIDI clips.

Note: this user manual covers only the “J74 Assistant Developers Tools” (this package). Readers interested in the “J74 MIDI Assistant” package are advised to look at that package separately.

Use Cases and Components

The “J74 Assistant Developers Tools” package provides connection points, code, rules, methods and example prompts for the use of an external “local-coding-enabled” AI Tool (such as Gemini CLI) as an assistant to Ableton Live. The toolset can also be used for automation purposes.

Package content and components:

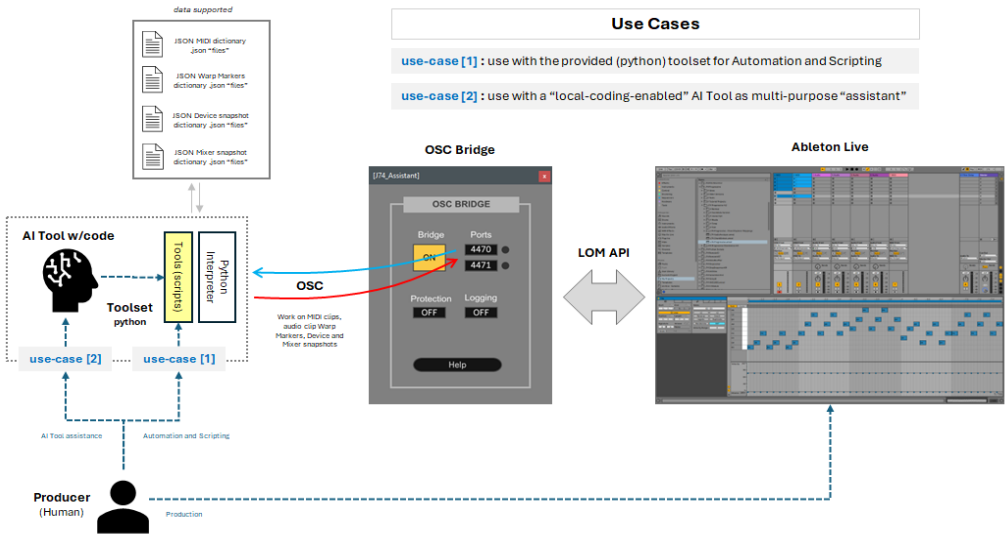
- An *OSC Bridge* (Max for Live) device, which is a transparent audio device for API communication via OSC. It performs logic control and packs and unpacks data for MIDI, audio warp markers and parameters.
- A *code repository* with a Python toolset of 30+ methods for communication, data formatting, data manipulation and structured API actions. The Python toolset alone addresses MIDI clip and Warp Markers manipulation, device snapshot and mixer snapshot as well as raw API calls.
- Part of the code repository are also work-flow descriptors, rules and methods for an AI Tool (Gemini CLI), items needed for making this AI directly capable of using the toolset and the OSC Bridge as a MCP (Model Context Protocol) local server for Ableton Live.
- A debugging facility for OSC communication of API calls (integrated in the Max for Live OSC Bridge).
- Documentation in PDF format (including this user manual)

As mentioned, two *use-cases* apply to this package:

1. The use of the provided Python toolset for automation and scripting of Ableton Live actions (snapshots, backups, MIDI operations, warp markers operations, etc.). This use does not involve an AI Tool.

2. The use of a “local-coding-enabled” AI Tool (such as Gemini CLI) as a direct-access, multipurpose assistant, capable of creating and manipulating an Ableton Live project directly (based the same components)

Below is a summary of the tools, functions and components in a diagram.

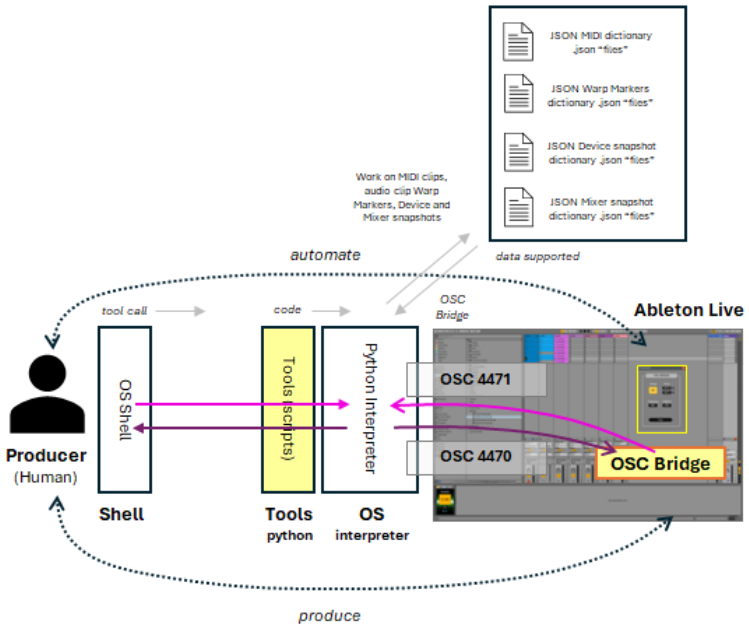


While working on an Ableton Live project, a producer can use these tools to [use-case1] automate backups, snapshots or large operations and [use-case2] for AI assistance on creative tasks such as: generation and processing of MIDI clips (using the AI background knowledge of harmony, melody, rhythm or even song structures); use of warp markers on audio clips (for instance adding swing to a sample); procedural creation of extremely large live sets; interpolation of device and mixer snapshots (and many others).

Next more details on each use case. Further, more on how to perform installation and setup.

Use case 1: Automation and Scripting of Ableton Live actions with Python and OSC

[1] *Automation and Scripting* of Ableton Live
(python scripts used directly by the user)



This use-case involves the following components:

- The OSC Bridge, which must be present in the live set. It is a transparent audio device and can be placed anywhere. In general, I would recommend placement on the Master track (as this track can never accidentally be deleted), but any other track would do too. The defaults on the OSC Bridge should cover all standard uses. More on the additional controls in the OSC Bridge later in this manual.

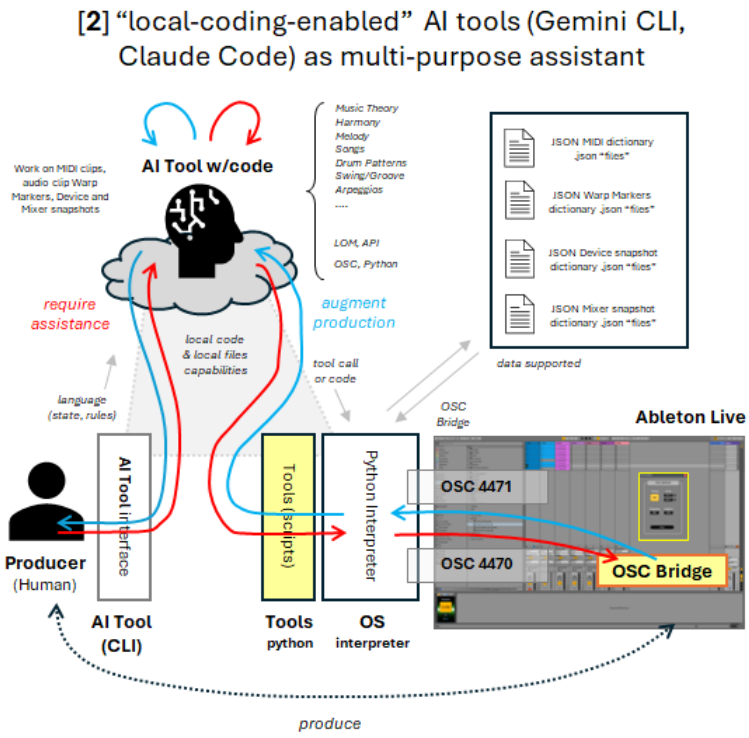
- The Python toolset provided with the package. This toolset offers classes and methods for accessing the API via OSC, with scripts covering all major operations. More on the Python toolset later in this manual.

The work-flow:

- The user (producer) calls Python scripts in the toolset directly from a Shell terminal to do actions. These tools can be automated via the OS "crontab" features (e.g. in Windows "taskschd.msc") or run manually on demand by the user (producer).

- The Python scripts operate through the OSC Bridge on the LOM API of Ableton Live.

Use case 2: Use of a "local-coding-enabled" AI Tool as a multi-purpose "assistant"



This use-case involves the following components:

- The OSC Bridge, which must be present in the live set. It is a transparent audio device and can be placed anywhere. In general, I would recommend placement on the Master track (as this track can never accidentally be deleted), but any other track would do too. The defaults on the OSC Bridge should cover all standard uses. More on the additional controls in the OSC Bridge later in this manual.

- The Python toolset provided with the package. This toolset offers classes and methods for accessing the API via OSC and a toolset of scripts which cover all major operations. This is the interface the "local-coding-enabled" AI Tool uses as its API (MCP service). More on the Python toolset later in this manual.

- A "local-coding-enabled" AI Tool such as Gemini CLI or Claude Code. These AI Tools can run or even create code on your computer and can read and write files on it. The AI is instructed by the environment to use the tools and eventually create additional code conform tool specifications. Note: this user manual only covers setup and workflow for one specific AI Tool: *Gemini CLI*. More on that later in this manual.

The work-flow:

- The user (producer) interacts with the AI Tool using common language, requiring the assistant for a given task ("Action: Create three different progressions in Eb minor in track 1 slots 1 to 3, using sparsely chord extensions." OR "Action: Create 64 Audio Tracks; set volume to -3dB on each of these, with per track an alternative full left and full right panning"). More on prompt structure and its possibilities later.

- The AI interprets the request with the foundational mandates and work-flows, which the AI gets as initial start-up with the provided environment (as a form of training), and translates these tasks into calls to the toolset or, if these calls are not sufficient for the task, the AI generates new code based on the environment rules and specifications provided.

- The Python scripts operate through the OSC Bridge on the LOM API of Ableton Live.

Installation and Setup

This chapter provides the instructions required to get the "J74 Assistant Developers Tools" components to work.

Note: this version of the procedure has minimal remarks and can be read quickly. If you prefer a more detailed guide, with additional explanation on each step, you might prefer the detailed version reported in Appendix [1].

Follow the steps carefully. Move on to the next step only if you are sure the previous step was completed.

[Step-1] – Download and Unzip the ZIP archive

To start *download* the package archive from the link you received after the purchase. Once the download has completed, *unzip* the download pack. Once unzipped, you will see the following content:

- The *OSC Bridge*, a Max for Live device plug-in named: "Assistant_Bridge_J74.amxd"
- The *J74 Assistant code repository*, a folder named "J74_Assistant_Code"
- A set of *PDF documents*, including this user manual.

[Step-2] – Placement of the Max for Live "OSC Bridge" plug-in in the Ableton Live project

Save (copy) the OSC Bridge ("Assistant_Bridge_J74.amxd") anywhere on your system, in any location you like. When you need it in Ableton Live, just drag and drop it from the Ableton Live browser onto the selected track. You can load it and leave it there, as you do not need to configure anything on it (its default will do).

Tip: you can include the location of the folder where the "Assistant_Bridge_J74.amxd" is located directly in the Ableton Live browser, go to the "Places" section of the Ableton Live browser and select [+] Add Folder...

[Step-3] – Placement of the "J74 Assistant Code" repository (folder) on your system

The J74 Assistant code repository is the folder named "J74_Assistant_Code" which you unpacked in the previous steps. Decide for a place on your computer drives where you wish to keep it. Copy it there. You can do it with the usual "File Explorer" (on Windows) / "Finder" (on a MAC) copy & paste actions or use the OS copy utilities below.

On a Windows system you can use the utility "robocopy" for copying all its content to the destination place:

```
robocopy "[SOURCE_PATH]\J74_Assistant_Code" "[DESTINATION_PATH]\J74_Assistant_Code" /E /Z /MT:16
```

On a MAC OS system you can use the utility "rsync" for copying all its content to the destination place:

```
rsync -avzhP --inplace "[SOURCE_PATH]/J74_Assistant_Code" "[DESTINATION_PATH]/J74_Assistant_Code"
```

IMPORTANT: please note down the location of the folder assigned to the code repository. *You will need that later.*

[Step-4] – Python Installation

The code repository consists of a set of Python scripts and classes made to work with the J74 OSC Bridge plug-in, plus configuration items for the AI Tool (Gemini CLI). It requires a Python interpreter to be installed on the OS.

MAC OS: On the MAC system Python is already per-installed with the system, but it can be an older version (you are advice to use a Python version which is 3.11 or newer). Therefore, after checking the pre-installed version (see next) we might need to install a newer version from https://www.python.org/

Windows: On a Windows system it is an add-on feature anyway, so we will typically have (unless you installed python before) to install it using the installer from https://www.python.org/.

*Tip: you can check if the python interpreter is already present on your system. Open a shell (command prompt) and type "*python --version*". If python is installed, you should see the version running (example "Python 3.13.13")*

ACTION: If you need to install python on your system, you should follow the instructions from https://www.python.org/.

Other remarks:

- MAC OS: If you keep the OS version of python, run all the scripts using the "python" inline. If you instead decide to install the newer version, use the "python3" inline command instead of "python".
- Windows OS: After installing Python run all the scripts using the "python" command

[Step-5] – The OSC library for Python ("python-osc")

Once you assured the Python interpreter is installed, you need to verify that the *"python-osc" library* is also present in Python. This library is a non-default library, and it is needed to let Python scripts "talk" OSC. You can check and eventually add the library directly from the shell prompt (Command Prompt).

On Windows and, if you use the system Python interpreter, on MAC OS use the "pip" utility of Python, in this way:

```
$$ pip install python-osc
```

Or, if you are on a MAC OS system but you installed the newest version, do:

```
$$ pip3 install python-osc
```

This completes the set-up for the Python part. Next, we will look at the set-up of the AI Tool, Gemini CLI.

[Step-6] – Install the "local-coding-enabled" Gemini CLI AI Tool

The following summarizes the major steps in getting Gemini CLI, the target AI Tool, to run on your computer.

Disclaimers:

- Gemini CLI is an AI Tool from *Google*. The "J74 Assistant Developers Tools" and its maker has no direct affiliation whatsoever with Google or any other AI platform. Gemini CLI has been chosen as a reference AI Tool due to its *generous free tier* and of course *for its "local-coding-enabled" capabilities*.
- To run Gemini CLI a (free) Google account (as with GMAIL) is required. Be sure you have a Google account before starting this step. When required (e.g. at the first launch) you will be asked to log in to that account to get the Gemini CLI AI Tool to run. Please make sure the log-in works.
- An AI Tool like Gemini CLI does NOT run entirely locally on your computer. It has an interface to your computer locally (for the prompts, for the code and for its file usage) but it also talks to the Google back-end computing data-centers via the Internet, where all its "intelligence" (AI models, neural networks and data) resides. Therefore, you MUST have your computer connected to the Internet to be able to use Gemini CLI.

Before starting you need to be aware of the following:

- *Gemini CLI runs on top of Node.js*, so this means that you will need to install Node.js first.
- Node.js packages and instructions can be found here: https://nodejs.org/en/download

- Gemini CLI instructions can be found here: https://geminicli.com/docs/get-started/installation/

IMPORTANT: Node.js is OS dependent and requires you to follow instructions carefully, as explained next.

- On Windows, Node.js supports the same installers on both Windows 10 and Windows 11. As far as I have seen, installation with the standard Node.js installer worked well on all the Windows systems I tested

- On MAC OS it depends on the OS version you run. At the time of writing, the following applies:
  - *On MAC OS 14 (Sonoma) and newer*, the generic installer (.pkg) provided by the Node.js site for the latest Node.js version 24 should work. Please use that installer as your primary target. Only if that fails fall back to the procedure described in Appendix [2] of this user manual
  - *On older versions of MAC OS, from MAC OS 13 (Ventura) down to MAC OS 11 (Big Sur)*, you need to install an older version of Node.js (such as version 22, a version which still satisfies Google CLI requirements). In this case perform the procedure described in Appendix [2] of this user manual.

ACTION: As of the time of writing, refer to the latest instructions from Node.js here:
https://nodejs.org/en/download

After Node.js is installed, follow the instructions provided by the Gemini CLI site:

ACTION: As of the time of writing, refer to the latest instructions from Google here:
https://geminicli.com/docs/get-started/installation/

Again, follow these instructions carefully and control the requirements before installation.

[Step-7] – Start Gemini CLI (log-in, trust of the code repository folder)

Once the installation process for Gemini CLI (and before that, Node.js) has been completed*, navigate to the folder you assigned to the code repository* (the location you have chosen in [Step-3], where you copied the code repository) and type "***gemini***".

```
D:\>cd "D:\My Projects\J74 Assistant\J74_Assistant_Code"
D:\My Projects\J74 Assistant\J74_Assistant_Code> gemini
```

The first time you do this Gemini CLI will require you to:

- *Trust of the selected "J74_Assistant_Code" code repository folder*
- *Authenticate with your Google account*

You need to do both. After logging in and trusting the folder, you should see something like this:

```
Gemini CLI is restarting to apply the trust changes...
 ***     Gemini CLI v0.38.2
   ***
  ***    Signed in with Google /auth
 **      Plan: Gemini Code Assist for individuals /upgrade
Logged in with Google: xxxxxxxx@gmail.com /auth
```

[Step-8] – Verifications for Gemini CLI and for the environment

You can quickly verify that the set-up for Gemini CLI has worked by doing the following in Gemini CLI:

> /memory show

```
ℹ Current memory content from 1 file(s):
  --- Project ---
  --- Context from: g:/J74_Assistant_Code/gemini.md ---
  <!-- Imported from: ./docs/osc-bridge-guidelines.md -->
  # GEMINI.md - Ableton Live OSC Bridge Workflow

This file defines the foundational mandates and workflows for the J74 OSC Bridge environment. These
instructions take absolute precedence over general defaults.
....
```

If that (see above snippet) is what you get, you can now proceed performing a *connectivity test* to the OSC Bridge in Ableton (of course, please make sure the OSC Bridge is running in Ableton Live *before* you start this test).

From the command line of Gemini CLI issue this *prompt* (you do not need to copy the leading ">"):

> Action: verify connectivity with Ableton Live

If the Gemini CLI AI Tool has loaded the environment properly, it will understand and automatically run a test Python script which tests both Max for Live and Python assets. If that is working, this will not only ensure connectivity but also the proper recognition and functioning of the entire Python environment in the AI Tool.

IMPORTANT: Gemini CLI will sometimes require permission to write and read files and for running code. This is normal. When asked, grant the permission using the "**Allow for this session**" option. In theory you can avoid this using the Gemini CLI "YOLO" ("you only live once") option, but that option is not recommended (for your computer safety) as it might allow Gemini CLI too much freedom on your computer. My recommendation is never to use the YOLO option in Gemini CLI (trust me, it did cause me headaches...).

Next as a simple API/OSC test, ask the AI to mute / unmute a track in the current live set, with a *prompt* like this:

> Action: mute track 1

And then:

> Action: unmute track 1

Finally, as a more advanced test, ask the AI Tool to build a MIDI clip (in this example track 1 is a MIDI or instrument track – if that is not the case, please change the prompt below accordingly). The *prompt* should be something like:

> Action: create a 1 5 6 4 MIDI progression in the C Major key in track 1 slot 1

If all that is working for you have done all the setup correctly!!!

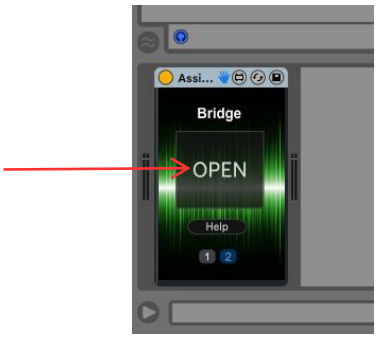
From now on you can effectively use the package to its full extent and work with your AI Tool assistant (Google CLI) on any Ableton Live project. Please read the next paragraphs to learn how to use the toolset and how to make efficient prompts.

In case of set-up and installation issues, please report those via email to: info [at] fabriziopoce.com

The OSC Bridge (its user interface and its parameters)

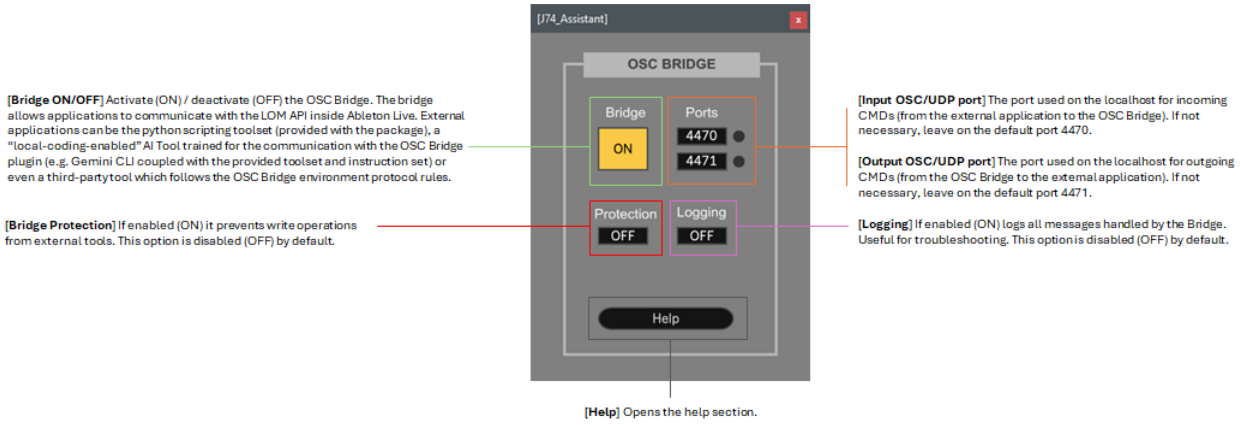
The Max for Live OSC Bridge is a “set and forget” piece of software. You load it onto your Ableton Live project and there is nothing you need to do on it. It will act as an OSC Bridge, as described.

Once dropped in the live set (on the Master track as by recommendation or anywhere else if you wish), the “Assistant_Bridge_J74.amxd” plug-in will appear as a small “docked” device, as in the picture below:



To see its complete user interface, for additional detail and for the log facilities, you can hit the “OPEN” button. If you find the user interface too small, select “2” on the [1|2] control to magnify the user interface.

Below a summary of the controls for the OSC Bridge section of the plug-in (once the user interface is open).



Note: In general, the defaults of the OSC Bridge are enough for it to work well. You might need to open its user interface only if you need to troubleshoot your environment (as it controls the logging facility).

Next:

- If you are interested in both [use-case1] (= the use of the Python toolset for automation) and [use-case2] (= the use of a “local-coding-enabled” AI Tool) read the next paragraphs in sequence.
- If you are interested primarily in [use-case2] (= the use of a “local-coding-enabled” AI Tool) you can jump to the “AI Tool Prompt engineering and prompt examples (for Gemini CLI)”. You can eventually go back and read about the Python toolset at a later stage.

The Python Toolset

The Python toolset provided with the package provides Shell (command line) scripts for a variety of operations. You can use these scripts for automation tasks; an external AI Tool will use these scripts and its MCP / API interface.

The "**core**" methods (scripts in the core/ folder) are the basic building blocks for interaction (through the OSC Bridge) with Ableton Live MIDI clips, audio clips (warp markers), devices and mixer.

Here a list of the "core" methods (scripts):

```
python core/clear_warp_markers.py <TRACK_NUMBER> <CLIP_NUMBER> {OPTION}
python core/collect_midi_clip.py <TRACK_NUMBER> <CLIP_NUMBER>
python core/collect_warp_markers.py <TRACK_NUMBER> <CLIP_NUMBER> {OPTION}
python core/create_midi_clip.py <TRACK> <SLOT> <JSON>
python core/create_warp_markers.py <TRACK_NUMBER> <CLIP_NUMBER> <JSON>
python core/device_snapshot.py <TRACK_NUMBER> <DEVICE_NUMBER>
python core/mixer_snapshot.py <OPTION{1,2,3}>
python core/restore_device_snapshot.py <TRACK_NUMBER> <DEVICE_NUMBER> <JSON>
python core/restore_mixer_snapshot.py <JSON> <OPTION>
```

A special "core" method is the "lom_command_osc.py" which is a script that transparently pipes a LOM API request (call) into Live and gathers the output of the response.

```
python core/lom_command_osc.py <PATH> <ACTION> <PROPERTY> {VALUE}
```

The "**extra**" methods (scripts in the extra/ folder and its sub-folders) provide specific additional tools for generation and manipulation services. They also use the same framework and work through the OSC Bridge.

Here a list of the "extra" methods (scripts):

```
python extra/device/inspect_device_params.py <TRACK_NUMBER> <DEVICE_NUMBER>
python extra/device/interpolate_device_snapshots.py <TRACK> <DEVICE_NUMBER> <JSON1> <JSON2> <WEIGHT>
python extra/device/randomize_device_params.py <TRACK> <DEVICE_NUMBER>
python extra/midi_clip_op/arpeggiate_clip.py <TRACK_NUMBER> <CLIP_NUMBER> <INTERVAL>
python extra/midi_clip_op/list_clip_notes.py <TRACK_NUMBER> <CLIP_NUMBER>
python extra/midi_clip_op/transpose_clip.py <TRACK_NUMBER> <CLIP_NUMBER> <SEMITONES>
python extra/midi_clip_op/transpose_conditional.py <TRACK_NUMBER> <CLIP_NUMBER> <PITCH> <CONDITION>
python extra/midi_json_gen/generate_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>
python extra/midi_json_gen/generate_melody_prob.py <ROOT> <SCALE> <PROG> <INTERVAL> <PROB> <OPTION>
python extra/midi_json_gen/generate_progression.py <ROOT> <SCALE> <PROGRESSION>
python extra/midi_json_gen/generate_progression_with_melody.py <ROOT> <SCALE> <PROG> <INTERV> <RAND>
python extra/midi_json_op/arpeggiate_midi.py <JSON> <INTERVAL>
python extra/midi_json_op/modify_midi_prob.py <JSON> <PROBABILITY> <OPTION>
python extra/midi_json_op/modify_midi_timing.py <JSON> <DELAY> <OPTION1> <OPTION2>
python extra/midi_json_op/modify_midi_vdev.py <JSON> <VDEV_VALUE>
python extra/midi_json_tool/midi_analyze.py <JSON>
python extra/mixer/randomize_mixer_settings.py <OPTION{1,2,3}>
python extra/mixer/interpolate_mixer_snapshots.py <JSON1> <JSON2> <WEIGHT>
python extra/mixer/inspect_mixer_settings.py
python extra/warp_clip_gen/generate_swing_markers.py <TRACK_NUM> <CLIP_NUM> <STEP_LEN> <AMOUNT>
python extra/warp_clip_op/move_warp_marker_relative_distance.py <TRACK_NUM> <CLIP_NUM> <PERCENT>
python extra/warp_clip_op/remove_markers.py <TRACK_NUMBER> <CLIP_NUMBER> <LIST_MARKERS>
```

Remarks:

- A short help for each script is provided running the script *with NO inline parameter*.
- A printout of the help for each script is provided in the "Ableton_OSC_Bridge" slide deck, which is an additional piece of documentation included in the package.

Examples of usage for the Python Toolset

The best way to familiarize yourself with the Python toolset is to run some of the scripts and see them in action with Ableton Live running. Remember to have the OSC Bridge placed somewhere in Ableton Live (on the Master track or any other track) before using these scripts.

So, let's log into the Shell (Command Prompt) and navigate to the working folder chosen for the code repository.

From this directory you can run any of the script.

Let's start with an example, using one of the core methods

Let's say I wish to collect a snapshot of the Mixer settings. Once I navigated to the folder, I first run the script without parameters, so to see its inline help:

```
C:\Windows\SysWOW64>G:
G:\>cd J74_Assistant_Code
G:\J74_Assistant_Code>python core/mixer_snapshot.py
-----------------------------------------------------------------------------------------------------
Usage: python core/mixer_snapshot.py <OPTION{1,2,3}>
-----------------------------------------------------------------------------------------------------
<OPTION> = 1 >>>> collect all tracks
<OPTION> = 2 >>>> collect only regular tracks
<OPTION> = 3 >>>> collect only return tracks
<OPTION> = 4 >>>> collect only the master track
-----------------------------------------------------------------------------------------------------
Example: python core/mixer_snapshot.py 1
-----------------------------------------------------------------------------------------------------
This method collects a mixer snapshot and saves it to a JSON file.
```

As you can see, the script requires an <OPTION> parameter. Let's choose option 1 (all tracks) and run it:

```
G:\J74_Assistant_Code>python core/mixer_snapshot.py 1
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Snapshot exported to data\Mixer_snapshot_20260421_1710.json
```

The script successfully ran and collected a Mixer snapshot (in the data/ folder).

Now go to Ableton Live and change whatever parameter you like on the mixer (volume, panning, send levels, mute, solo, etc.). Let's see the use of the "restore_mixer_snapshot.py" script to restore the snapshot we just collected.

First, just to show it to you, I run the script without parameters:

```
G:\J74_Assistant_Code>python core/restore_mixer_snapshot.py

-----------------------------------------------------------------------------------------------------

Usage: python core/restore_mixer_snapshot.py <JSON> <OPTION>
-----------------------------------------------------------------------------------------------------
<JSON> : The JSON file for the snapshot to restore.
<OPTION> : use 1 to force the restore operation even if the track number is different from the project.
-----------------------------------------------------------------------------------------------------
Example: python core/restore_mixer_snapshot.py data/Mixer_snapshot_20260416_0945.json 0
-----------------------------------------------------------------------------------------------------
This method sends a mixer snapshot from a JSON file to the mixer.
```

Now we run the “restore_mixer_snapshot.py” using the snapshot we just collected:

```
G:\J74_Assistant_Code>python core/restore_mixer_snapshot.py data\Mixer_snapshot_20260421_1710.json
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Restore Complete!
```

Another example this time with a device. Place a device of choice in a track of choice. In my example I will do track 3 and the device will be the second one on the track (it is an “EQ Three” device).

Let’s collect a snapshot of this device with the “device_snapshot.py” script (I will omit the help output this time):

```
G:\J74_Assistant_Code>python core/device_snapshot.py 3 2
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Collecting snapshot for Track 2, Device 1...
Accessing device at live_set tracks 2 devices 1...
Device Name: EQ Three
Successfully saved snapshot to data\EQ_Three_snapshot_20260421_1720.json
```

As a similar test I will change the device settings in Ableton Live and then restore it with the “restore_device_snapshot.py” script:

```
G:\J74_Assistant_Code>python core/restore_device_snapshot.py 3 2 data\EQ_Three_snapshot_20260421_1720.json
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Source device class (from metadata): EQ Three
Checking target device at live_set tracks 2 devices 1...
Target device class_display_name: 'EQ Three'
Match found! Proceeding with snapshot push...
Snapshot push complete.
```

Let’s move on to MIDI clips.

Suppose you have a MIDI clip already in the set and you want to collect its content. In my example it will be in track 1 clip slot 1. We will use the “collect_midi_clip.py” script for that. First with no parameters to show the help:

```
G:\J74_Assistant_Code>python core/collect_midi_clip.py
-----------------------------------------------------------------------------------------------------
Usage: python core/collect_midi_clip.py <TRACK_NUMBER> <CLIP_NUMBER> {OPTION}
-----------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
{OPTION} : Use 1 for printing the JSON output.
-----------------------------------------------------------------------------------------------------
Example: python core/collect_midi_clip.py 1 1
-----------------------------------------------------------------------------------------------------
This method collects notes from a MIDI clip and saves them to a JSON file.
```

Then the real collection:

```
G:\J74_Assistant_Code>python core/collect_midi_clip.py 1 1
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Requesting extended notes for live_set tracks 0 clip_slots 0 clip...
Waiting for /requested_notes response (collecting multiple notes)...
[INFO] Silence timeout reached. Assuming all notes collected.
>>> SUCCESS: Collected 5 notes.
[INFO] Notes saved to data\MIDI_Clip_1_1_20260421_1724.json in LOM format.
```

Now, for a test, let's manually delete that clip in Ableton Live.

Let's use "create_midi_clip.py" to send the collected MIDI notes back to Ableton Live (a new clip will be created):

```
G:\J74_Assistant_Code>python core/create_midi_clip.py 1 1 data\MIDI_Clip_1_1_20260421_1724.json
[INFO] 'clip_length' not found in JSON. Calculated as 4.0
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Creating new clip with length 4.0...
Adding 5 notes to Track 1, Slot 1...
Successfully created MIDI clip on track 1 clip 1 from LOM format JSON.
```

This should be straight forward at this point. In essence all the "core" methods provide the basic operations for collecting and for creating things (MIDI clip with notes, warp markers in audio clips and device and mixer snapshots).

Let's look at a little funnier set of examples using the "extra" methods.

Let's create a brand-new MIDI clip with some chord progression of triads. One script has been created for that, the "generate_progression.py". We will use it. Again, from the same location, I run this script with no parameter to see its inline help:

```
G:\J74_Assistant_Code>python extra/midi_json_gen/generate_progression.py
-----------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_gen/generate_progression.py <ROOT> <SCALE> <PROGRESSION>
-----------------------------------------------------------------------------------------------------
<ROOT> : The root of the selected scale, as a letter, like G in {G Minor}
<SCALE> : The root of the selected scale, as a letter, like Minor in {G Minor}.
<PROGRESSION> : The progression expressed as a list of degrees such as [1,5,6,4].
-----------------------------------------------------------------------------------------------------
Example: python extra/midi_json_gen/generate_progression.py G Minor [1,5,6,4]
-----------------------------------------------------------------------------------------------------
This method creates a MIDI clip with a given progression, as straight chords.
```

The script requires a Root (example “G”), a Scale (example “Minor”) and a list of degrees (like [1,5,6,4]) like it suggests in the Example. Let’s use exactly that example:

```
G:\J74_Assistant_Code>python extra/midi_json_gen/generate_progression.py G Minor [1,5,6,4]
Generated data/MIDI_Clip_G_Minor_prog_20260421_1733.json
```

As you can see the script created a JSON data set. Let’s push this data set to Ableton Live, this time on an unused clip slot. In my example track1 slot 2. We will use the same “create_midi_clip.py” script we used before:

```
G:\J74_Assistant_Code>python core/create_midi_clip.py 1 2 data\MIDI_Clip_G_Minor_prog_20260421_1733.json
[INFO] 'clip_length' not found in JSON. Calculated as 16.0
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Creating new clip with length 16.0...
Adding 12 notes to Track 1, Slot 2...
Successfully created MIDI clip on track 1 clip 2 from LOM format JSON.
```

Next, an example with warp markers. Let’s load a drum loop on an audio clip (assumption: warp is on for audio clips, this clip is looped and it is a sort of “straight timing” drum loop) and let’s use the following “extra” method to create some swing using warp markers. The clip in my example is on track 3, slot 1. I will ask for a swing amount of 0.4 (40%, delayed as this amount is positive) and a base of 1/16th intervals (that is a beat interval of 0.25). The script automatically generation warp markers and moves off-beat transients as 40% delayed:

```
G:\Gemini_Project>python extra/warp_clip_gen/generate_swing_markers.py 3 1 0.25 0.4
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Collecting markers for live_set tracks 2 clip_slots 0 clip...
Cleanup complete.

[INFO] Requesting clip information for live_set tracks 2 clip_slots 0 clip...
Generated 33 markers, saved to data\Warp_Markers_Modified_20260423_1106.json
[INFO] Clip length: 8.0
Adding 1 warp markers from data\Warp_Markers_Modified_20260423_1106.json...
Adding Marker: Beat 0.0000, Sample 0.0000
Adding Marker: Beat 0.3500, Sample 0.1500
Adding Marker: Beat 0.5000, Sample 0.3000
....
Adding Marker: Beat 8.0000, Sample 4.8000
Action complete.
```

Next another MIDI experiment. Let’s generate a random melody in a given key and within a set of chords. This melody needs to have some randomness by using the “probability” feature in Live (so if it loops, it will loop each time in a slightly different way). This is ideal to have some sort of evolving sequence on top of a chord progression.

```
G:\Gemini_Project>python extra/midi_json_gen/generate_melody_prob.py G Minor [1,5,6,4] 0.25 0.5 1
Generated data/MIDI_Clip_G_Minor_melody_prob_20260423_1113.json

G:\Gemini_Project>python core/create_midi_clip.py 1 7 data/MIDI_Clip_G_Minor_melody_prob_20260423_1113.json
```

```
[INFO] 'clip_length' not found in JSON. Calculated as 16.0
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Creating new clip with length 16.0...
Adding 64 notes to Track 1, Slot 7...
Successfully created MIDI clip on track 1 clip 7 from LOM format JSON.
```

As you can see, we first use the “python extra/midi_json_gen/generate_melody_prob.py” to generate the JSON MIDI data and the push this data to Ableton Live on track 1 slot 7 with the “core/create_midi_clip.py” method.

Next let’s do some processing. Let’s say we want to take a MIDI clip from Live and modify its timing with some swing-like effect as a new MIDI clip. For this we split the action in tasks: [a] we get the MIDI clip from Ableton Live with the “core/create_midi_clip.py” (this will create a JSON file), [b] we then modify the MIDI data using the “extra/midi_json_op/modify_midi_timing.py” script (using the options for modifying only off-beat notes and for random movement – delay or anticipate) and finally [c] we upload the resulting MIDI JSON data to another MIDI clip in Ableton live.

```
G:\Gemini_Project>python core/collect_midi_clip.py 1 7
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Requesting extended notes for live_set tracks 0 clip_slots 6 clip...
Waiting for /requested_notes response (collecting multiple notes)...
[INFO] Silence timeout reached. Assuming all notes collected.
>>> SUCCESS: Collected 64 notes.
[INFO] Notes saved to data\MIDI_Clip_1_7_20260423_1123.json in LOM format.

G:\Gemini_Project>python extra/midi_json_op/modify_midi_timing.py data/MIDI_Clip_1_7_202604_1123.json 0.25 1 2
data/MIDI_Clip_Modified_202604_1128.json

G:\Gemini_Project>python core/create_midi_clip.py 1 8 data/MIDI_Clip_Modified_202604_1128.json
[INFO] 'clip_length' not found in JSON. Calculated as 16.0
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
[INFO] Creating new clip with length 16.0...
Adding 64 notes to Track 1, Slot 8...
Successfully created MIDI clip on track 1 clip 8 from LOM format JSON.
```

Here is another interesting method, a script which can tell you in which musical key a MIDI clip (after you collected it as a JSON data file) is into.

```
G:\Gemini_Project>python extra\midi_json_tool\midi_analyze.py
-----------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_tool/midi_analyze.py <JSON>
-----------------------------------------------------------------------------------------------------
<JSON> : The JSON file containing the MIDI note original information.
-----------------------------------------------------------------------------------------------------
Example: python extra/midi_json_tool/midi_analyze.py data/MIDI_Clip_G_Minor_prog_20260416_1200.json
-----------------------------------------------------------------------------------------------------
```

```
This method retrieves notes from a JSON file and analyzes it, providing if possible a matching musical key.


G:\Gemini_Project>python extra/midi_json_tool/midi_analyze.py data/MIDI_Clip_Modified_20260423_1128.json
Possible scales (alternatives with the same note set):
C Dorian
D Phrygian
D# Lydian
F Mixolydian
G Minor
A Locrian
A# Major

Note set:
[0, 2, 3, 5, 7, 9, 10]

Note names:
['C', 'D', 'D#', 'F', 'G', 'A', 'A#']
```

Another useful set of methods are the “interpolation” methods.

There is one for mixer settings and one for device settings. They take two JSON files (let’s say a JSON snapshot1 and a JSON snapshot2), they check if they are compatible (same device, same mixer set-up) and interpolate linearly between the two, given a weight (a percentage for the first versus the second, which you select).

Let’s try it on a device. For this we will use the method called “extra\device\interpolate_device_snapshots.py”

Let’s break this in sub-actions: [a] collect a device snapshot from a device in live [b] modify the same device settings in Live (as you like) and collect a second device snapshot and finally [c] perform the interpolation (with a 50% weight among the two snapshots) and push this new interpolated snapshot to the device in Live.

```
G:\Gemini_Project>python core\device_snapshot.py 3 1
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Collecting snapshot for Track 2, Device 0...
Accessing device at live_set tracks 2 devices 0...
Device Name: EQ Three
Successfully saved snapshot to data\EQ_Three_snapshot_20260423_1144.json

G:\Gemini_Project>python core\device_snapshot.py 3 1
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Collecting snapshot for Track 2, Device 0...
Accessing device at live_set tracks 2 devices 0...
Device Name: EQ Three
Successfully saved snapshot to data\EQ_Three_snapshot_20260423_1145.json

G:\Gemini_Project>python extra\device\interpolate_device_snapshots.py
```

```
------------------------------------------------------------------------------------------------------------

Usage: python extra/device/interpolate_device_snapshots.py <TRACK> <DEVICE_NUMBER> <JSON1> <JSON2> <WEIGHT>

------------------------------------------------------------------------------------------------------------

<TRACK_NUMBER> : The track where the target clip is located

<DEVICE_NUMBER> : The device number in the track (counting from left).

<JSON1> : The first JSON file for the interpolation.

<JSON2> : The second JSON file for the interpolation.

<WEIGHT> : The weight for the interpolation from 0 to 1.

------------------------------------------------------------------------------------------------------------

Example: python extra/device/interpolate_device_snapshots.py 3 1 data/Auto_Filter_snapshot_20260416_0939.json
data/Auto_Filter_snapshot_20260416_0958.json 0.33

------------------------------------------------------------------------------------------------------------

This method interpolates two device snapshots from JSON files and send the interpolation to a device.


G:\Gemini_Project>python extra\device\interpolate_device_snapshots.py 3 1
data\EQ_Three_snapshot_20260423_1144.json data\EQ_Three_snapshot_20260423_1145.json 0.5

OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471

  Setting [0] Device On to 1.0

  Setting [1] GainLo to 0.48452382162213326

  Setting [2] GainMid to 0.925000011920929

  Setting [3] GainHi to 0.9210317432880402

  Setting [4] FreqLo to 0.3850599676370621

  Setting [5] FreqHi to 0.3758864775300026

  Setting [6] LowOn to 1.0

  Setting [7] MidOn to 1.0

  Setting [8] HighOn to 1.0

  Setting [9] Slope to 1.0


Interpolation and Push complete!
```

As you can see there is potential here. In case of devices, you can create new presets merging existing ones or you can backup / A-B-evaluate settings. And for mixing go back to a previous mix without reloading the project (and eventually use other edits).

A final mention goes the “raw” LOM API transparent call, the “core/lom_command_osc.py” script. This is not a script you will often run “alone”, but it is a script you can use in other scripts (as it is a building block). The AI Tool will use this extensively for your required actions. This script just does “API calls via OSC”. You define the API “path”, the type of “action” (get, set, call), the “property” (target). If applicable you give as last the “value”.

```
G:\Gemini_Project>python core/lom_command_osc.py

Usage: python tmp/lom_command_osc_cli.py <path> <action_type> <property> [<value>]

Example (Get): python tmp/lom_command_osc_cli.py "live_set tracks 0" get name

Example (Set): python tmp/lom_command_osc_cli.py "live_set tracks 0" set name "New Name"

Example (Call): python tmp/lom_command_osc_cli.py "live_set tracks 0" call stop_all_clips
```

```
G:\Gemini_Project>python core/lom_command_osc.py "live_set tracks 0" call stop_all_clips
OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
Sending LOM Command:
  Path:        live_set tracks 0
  Action:      call
  Property:    stop_all_clips
[SUCCESS] Response received:
  Data: ['stop_all_clips', 'id', 0]
```

Tip: A printout of the help for all these Python scripts part of the “core” and “extra” methods is provided in the “Ableton_OSC_Bridge” slide deck, a piece of documentation included in the package. Please head there to see the possibilities offered by the Python toolset.

A final remark. The use of this Python toolset can be enhanced if you use something to schedule the script calls (e.g. a crontab feature, for instance to run a periodic backup of your mixer settings during mixing) or, for performing a large number of calls (e.g. creating a library of presets, interpolating across them).

This aspect (scheduling the Python scripts) is not covered by this user manual or by the environment itself, as it is highly personal and depends on what you want to achieve. Anyway the building blocks are there for you to use.

Next: AI Tool prompt engineering and usage is what we are going to look at next.

AI Tool Prompt engineering and prompt examples (for Gemini CLI)

Google's Gemini CLI is the ideal companion for the environment provided by the "J74 Assistant Developers Tools" as it is *free for limited usage*, with a generous *free quota* and it has all we need to let it assist on your producer projects (it can run shell / Python code and it can read and save files).

This AI knows also about music (harmony, melody, song structures, rhythm, drumming, timing, filtering – you name it), it knows about Ableton Live (what it is, how it works), it knows about the Ableton Live LOM API (every single command, attribute and detail) and it knows how to use the Python toolset of the "J74 Assistant Developers Tools" (it has been instructed to know by how the code repository is defined). And, of course, it understand the *common language* [*].

Note[*]: A funny note about prompts and languages. I am personally so used in doing everything technical in English, but that is not my native language. I am Italian, although nowadays I never use the Italian language for neither music nor technical work. So, I prompt everything I do with AI's in English. I am used to.

Actually, an AI has no nationality and no mother tongue either. An advanced AI Tool like Gemini CLI does perfectly understand prompts made in *any* common language. So you do not have to stick to English…

Here (for fun) an example in Italian:

```
************************************************************************************************************
> Azione: crea una progressione in La minore nella seconda traccia, posizione 2. La progressione e' 1,6 5,4
************************************************************************************************************
✦ Certamente. Procedo con la creazione della progressione in La minore (i-VI-v-iv) nella traccia 2, slot 2.

  Passaggio 1: Creazione della clip (4 battute)
  Utilizzo core/lom_command_osc.py per creare una clip vuota di 16 battiti (4 bar).
  LOM: live_set tracks 1 clips 1 (Traccia 2, Slot 2 in indici 0).

  Passaggio 2: Generazione e invio delle note
  Note per La minore (1-6-5-4):
   1. La minore (i): [45, 48, 52] | Inizio: 0.0, Fine: 4.0
   2. Fa Maggiore (VI): [41, 45, 48] | Inizio: 4.0, Fine: 8.0
   3. Mi minore (v): [40, 43, 47] | Inizio: 8.0, Fine: 12.0
   4. Re minore (iv): [38, 41, 45] | Inizio: 12.0, Fine: 16.0

************************************************************************************************************
* ✓  Shell python core/lom_command_osc.py "live_set tracks 1" call create_clip 16.0
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* Sending LOM Command:
*   Path:        live_set tracks 1
*   Action:      call
*   Property:    create_clip
*   Value:       16.0
* [INFO] Command sent. No specific return value received (common for 'call' actions).
************************************************************************************************************
* ✓  WriteFile  La_Minore_1654_progression.json → Accepted (+16, -0)
       1 {
       2   "notes": [
       3     {"pitch": 45, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       4     {"pitch": 48, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       5     {"pitch": 52, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       6     {"pitch": 41, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       7     {"pitch": 45, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       8     {"pitch": 48, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       9     {"pitch": 40, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
```

```
      10     {"pitch": 43, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      11     {"pitch": 47, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      12     {"pitch": 38, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      13     {"pitch": 41, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      14     {"pitch": 45, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0}
      15   ]
      16 }
*************************************************************************************************************
✦ Ora spingo le note nella clip utilizzando core/create_midi_clip.py (Traccia 2, Slot 2).
*************************************************************************************************************
* ✓  Shell python core/create_midi_clip.py 2 2 data/La_Minore_1654_progression.json
* [INFO] 'clip_length' not found in JSON. Calculated as 16.0
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* [INFO] Creating new clip with length 16.0...
* Adding 12 notes to Track 2, Slot 2...
* Successfully created MIDI clip on track 2 clip 2 from LOM format JSON.
*************************************************************************************************************
✦ La progressione 1-6-5-4 in La Minore (Am, F, Em, Dm) è stata creata con successo nella Traccia 2, Slot 2.
```

This just shows you the flexibility of an AI tool. This assistant is quite fluent in Italian too!

So, if you like, try to work with this AI with prompts in your native language. Yes, please go ahead and try. Just one remark from me: don't ask me to troubleshoot anything I cannot understand!

Some generic guidelines

Working with an AI it is always important to keep a few things in mind.

- An AI needs to know the "*context*" (what are you and the AI supposed to work on).
- An AI needs to know its "*role*" (and level of "expertise" desired) for interacting well with a human.
- Another important thing to take in mind is that an AI *needs guidance*.
  - You need to be *clear* (and not ambiguous).
  - You need to be *explicit* (if something is necessary, do not omit it).
  - You need to be *concise* (and not elaborate unnecessary - it will just consume tokens!)
  - *Don't assume the AI knows, if you are not sure the AI knows*.
  - Be *careful* in what you ask. Such a "local-coding-enabled" AI has a lot of power on your computer.

Think about the AI as a "*junior assistant producer*", someone that never says NO, who knows everything musically speaking (from a technical point of view) and knows Ableton Live too (but never made a tune...). In the case of this package someone who has also got the tools to work with it and is qualified to work. But as a "junior assistant producer", indeed an assistant and a junior, *it needs to be guided and guard-railed by you*.

Prompt "rules" for the J74 Assistant Developers Tools

Having the generic guidelines in mind, the environment of the "J74 Assistant Developers Tools" package fortunately addresses upfront many of these aspects, by crafting a work-flow with rules, methods and examples which is provided to the AI at the start-up of a session (automatically loaded by the AI in memory when it starts). This is exactly what the descriptors and the Python methods and data structure have been made for.

So, to work well with the AI (Gemini CLI), just follow this set of simple rules.

RULE 1: provide your Ableton Live action prompts using the "Action:" command:

> ***Action:*** *<formulate here your request in common language>*

RULE 2: For a short list of example prompts use the help command:

> ***Help:*** *<provide a keyword or category, such as: {Track, Mixer, Device, Parameter, MIDI, Audio}>*

Comments:

- The AI (Gemini CLI) has been instructed by the environment of the "J74 Assistant Developers Tools" that when these two types of prompts are given a specific context applies, with methods and rules. It knows.
- In the case of "*Action:*" prompts the AI knows that what you are requesting is an Ableton Live task, which needs to be evaluated against the environment rules and for which the (Python) toolset of methods and scripts need to be used. It will try to perform the task immediately after you entered this prompt. It may take time for the AI depending on the request. So, if you see "Thinking..." do *not* issue new prompts (which will interrupt the one still going) but wait for a response from the AI.
- We will go through many prompt examples in the next paragraphs, so do not worry about that too much now. From the examples you should see that the way you request things is very musical / producer-like language. It should be noticed though that information needs to be given in a relatively precise manner (some room for interpretation may be ok, but in general you should be precise). So, a prompt like "Action: Make me a hit song" it is not going to work. But then, would you, as a senior producer, ask your junior assistant for that kind of task?).

RULE 3: for any other conversational prompts (something which does not involve any task on the Ableton Live project) just use a conversational common language prompt (without any key-word like "Action:" or "Help:").

> ***What time is it dude?***

Comments:

- That last example is actually a bad idea for a prompt. It will consume tokens (quota) for something you can do otherwise, and you do not need an AI for. So, try to be efficient with your quota of tokens. Ask things to your AI assistant which are "useful" and that "save you time" or that "you cannot do without the assistant" (like "write me a jazz progression in a playing style xyz..."). Don't waste its tokens (time and resources).
- For many prompts it is maybe a good idea to use a Web Tool AI on the side. My suggestion is to use the "local-code-enabled" AI Tool assistant with the "J74 Assistant Developers Tools" for things which are Ableton Live tasks and use other AI tools for other things. It will save you tokens for when you need them.

Prompt Examples – MIDI clip note creation (harmony or melody)

> Action: create a midi clip in track 1 slot 1 with a chord progression IV V I VI in the G minor scale

> Action: create a midi clip in track 1 slot 1 with a progression I V IV II in C major 1 bar per chord

> Action: create a midi clip in track 1 slot 1 with the progression of the chorus of Taylor Swift "Cruel Summer" and on top

> Action: create a midi clip in track 1 slot 1 with the progression of the chorus of Queen "We are the champions" and on top

> Action: create a midi clip in track 1 slot 1 with a jazz progression being played with some swing and with increasing

> Action: create a midi clip in track 1 slot 1 with the progression of the chorus of Taylor Swift "Cruel Summer" and on top

> Action: create a midi clip in track 1 slot 1 with the progression of The Beatles "Let it be" chorus

> Action: create a midi clip in track 1 slot 1 with the progression of The Beatles "Let it be" chorus and add a melody on top

> Action: create a midi clip in track 1 slot 1 with the progression of a typical blues song, expanding chords up to 4 notes

> Action: create a midi clip in track 1 slot 1 with a chord progression IV V I VI in the Eb major scale with a random melody one octave higher using 1/8 notes

> Action: create four midi clips in track 1 slot 1 to 4 with four alternative chord progressions in the G minor scale each time with a random melody one octave higher using 1/8th notes. The melody notes should have probability 100% for downbeat notes and probability 33% for other notes.

Prompt Examples – MIDI clip note creation oriented to drumming.

> Action: create a midi clip on track 2 slot 1 with a drum beat. This drum beat should have a carribean flavor with sixteen instruments (in typical drum rack layout) make it 4 bars long and variate each bar.

> Action: create a midi clip on track 2 slot 2 with a drum beat (kick, snare, c_hat, op_hat). Kick on every 1/4th interval, snare on beat 2 and 4, o_hat is at 1/8th odd intervals and c_hats every 1/16th.

> Action: create a midi clip on track 2 slot 3 with a drum beat (kick, snare, closed_hat, open_hat). Kick is on each 1/4 interval, snare is at beat 1 and 3. Open_hat is at 1/8 odd intervals. For the closed_hats use the timing of the clip in track 3 slot 1.

Prompt Examples – MIDI clip note inspection

> Action: read the MIDI notes in the clip on track 1 slot 2 and list them as (pitch, time, velocity, mute) tuples

> Action: read the notes used by the clip on track 2 slot 1 and define in which western key these notes are. Assume the first note as root.

Prompt Examples – MIDI clip note modification of pitch

> Action: duplicate the clip in track 1 slot 1 (double length, repeat the clip notes)

> Action: modify the clip in track 1 slot 1 and transpose it a semitone up

> Action: modify the clip in track 1 slot 1 and transpose only C# and G# notes a semitone down

> Action: harmonize the clip 0 0 to the C minor key

Prompt Examples – MIDI clip note modification of time and number of notes

> Action: transform the clip in track 1 slot 1 into a progression where each chord's notes are arpeggiated sequentially at 1/4 bar intervals.

> Action: transform the clip in track 1 slot 1 into a progression where each chord's notes are arpeggiated sequentially at 1/4 bar intervals (repeat notes if necessary).

Prompt Examples – MIDI clip note modification with groove / swing timing in mind

> Action: transform the clip in track 1 slot 1 and randomize the timing of each note about 10% with exceptions of notes at beat 1 and 3)

> Action: Take the notes of track 1 clip 1 and modify the timing of the notes for non downbeat notes to be delayed of 1/32. Make the new clip in track 1 clip 5

Prompt Examples – MIDI clip note modification with probability /velocity deviation in mind

> Action: Take the notes of track 1 clip 3 and modify the probability of the notes for non downbeat notes only to be 50%. Make the new clip in track 1 clip 2

> Action: Take the notes of track 1 clip 2 and modify the velocity deviation of the notes for non downbeat notes only to be 50%. Make the new clip in track 1 clip 1

Prompt Examples – Warp Markers

> Action: Add warp markers every 1/16 of a bar in track 3 slot 3

> Action: move the "even" warp markers of track 3 slot 3 so to have these delayed of 1/32 of a bar

> Action: remove the 5th and 6th warp markers from clip slot 3 of track 3

> Action: change the warp mode of track 3 slot 3 to Re-Pitch

> Action: Clear the warp markers in track 3 clip 3

> Action: on track 3 slot 3 add warp markers every 1/16th and then move the non-downbeat warp markers so to have these delayed of 1/32th

Prompt Examples – Device Snapshots operations (collect, restore, interpolate, randomize)

> Action: collect a device snapshot for the first device on track 3

> Action: restore the device snapshot "EQ_Eight_snapshot_20260402_1250.json" on the second device on track 3

> Action: interpolate between device snapshots "EQ_Eight_snapshot_20260402_1222.json" (with weight 33%) and "EQ_Eight_snapshot_20260402_1250.json" (with weight 67%) and push the resulting values to the third device on track 3.

> Action: randomize the parameters of the fourth device in track 3

Prompt Examples – Mixer Snapshots operations (collect, restore, interpolate, randomize)

> Action: create a new mixer snapshot

> Action: restore the mixer snapshot Mixer_snapshot_20260402_1905.json

> Action: interpolate between mixer snapshots "Mixer_snapshot_20260402_1905.json" (with weight 25%) and "Mixer_snapshot_20260402_1906.json" (with weight 75%) and push the resulting values to the tracks.

> Action: randomize the mixer parameters for panning and send of the first fourth tracks

Prompt Examples – Actions which employ transparent (raw) LOM API calls on tracks

> Action: create 8 midi tracks from track 1

> Action: add 4 audio tracks after track 3

> Action: delete the first 4 tracks

Prompt Examples – Actions which employ transparent (raw) LOM API calls on mixer settings

> Action: mute track 3

> Action: unmute track 3

> Action: arm tracks 1 to 4

> Action: set the send A of track 1 to 0.5

> Action: set send A to 0.25 on the first 2 tracks and 0.5 on tracks 3 to 4

> Action: set alternative Pan full-left full-right on all tracks

Prompt Examples – Actions which employ transparent (raw) LOM API calls on specific device parameters

> Action: modify filter 3 on the first device in track 2 with q = 0.8 and frequency 3Khz

> Action: disable the second device in track 3

Prompt Examples – Actions which employ transparent (raw) LOM API calls on clips and clip slots

> Action: start the clip in track 1 slot 1

> Action: stop the clip in track 1 slot 1

> Action: delete the clip on track 1 slot 1

Prompt Examples – Actions which employ transparent (raw) LOM API calls on the Ableton Live application

> Action: check which version of Ableton Live is running

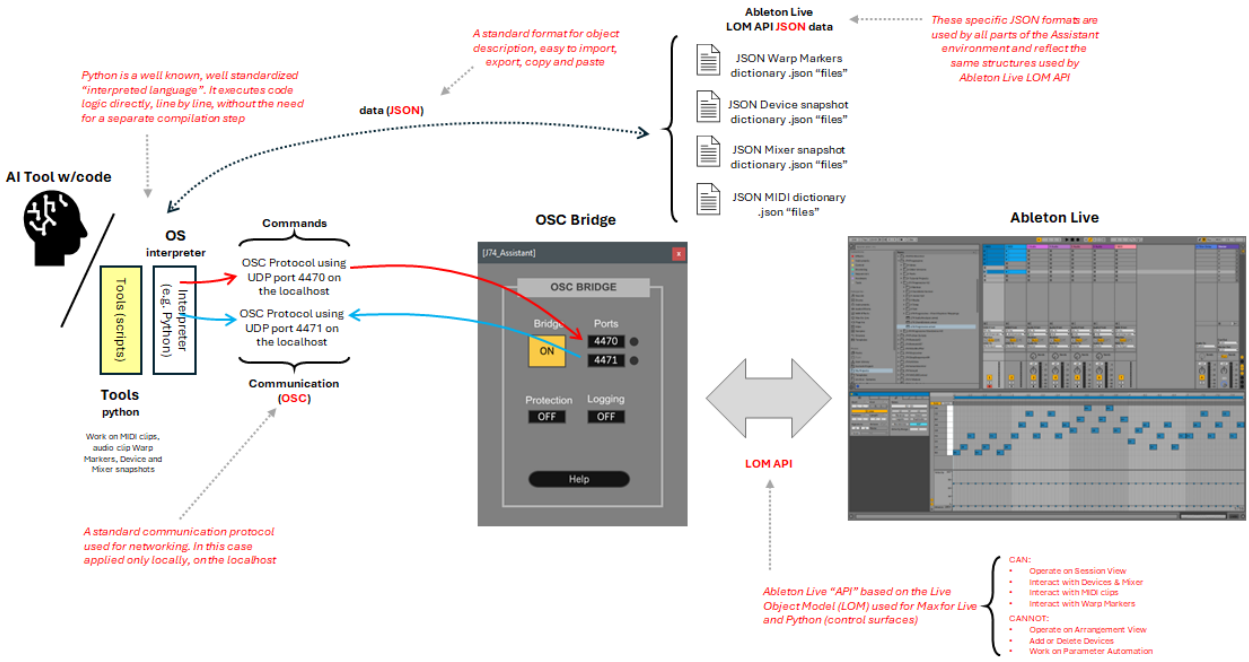
> Action: make a table for the mute state of tracks 1 to 6

> Action: make a list of all the send values for all the tracks

> Action: undo the last change

Overview of Data types and Standards used in this package

Here an overview of protocols, standards and data formats used by the “J74 Assistant Developers Tools” package.



You can use this diagram as a reference for understanding the architecture of the package. You can refer to this diagram when learning things.

Remarks on the working folder structure and content

Before we move on a quick look at the code repository file and folder structure. Once unpacked and copied to the folder you selected, the environment will look as below (a list of folders and sub-folders):

```
Directory of G:\XXXXXX
13-04-2026  08:59    <DIR>          .
13-04-2026  08:59    <DIR>          ..
15-04-2026  23:09    <DIR>          core
18-04-2026  12:33    <DIR>          data
15-04-2026  21:31    <DIR>          defs
07-04-2026  16:10    <DIR>          docs
16-04-2026  10:36    <DIR>          extra
07-04-2026  14:14    <DIR>          help
07-04-2026  16:17    <DIR>          init
20-04-2026  13:21    <DIR>          tmp
```

Here a brief explanation of what each folder contains, its purpose and way to use it:

- The "**core**/" folder: This is the folder which contains the "core" Python methods (scripts) which are both used for automation and by the AI Tool for handling actions. Consider this folder "read-only". Do not delete files and (unless you are brave) modify them (if you do, I will not be able to support your changes).

- The "**data**/" folder: This folder is a storage place for JSON data. It contains JSON data from older actions and examples. You can eventually clean it if you please, but you are advised not to as an AI Tool can "feed" on examples too. Think about it as the place where scripts dump and collect data. These files are miniscule, so do not worry too much about it growing too much.

- The "**defs**/" folder: This folder contains formal definitions of JSON Data Formats used by the toolset. Its main function is documenting the formats, and it is used by the AI environment. Consider this folder "read-only". Do not delete its content.

- The "**docs**/" folder: This folder contains a Gemini CLI descriptive file called "osc-bridge-guidelines.md". This file is a Gemini CLI specific descriptor, and it is quite important for getting the AI Tool to work well. Therefore, it MUST not be deleted. This file provides the AI Tool (Gemini CLI) with all the rules, logic and methods it needs to be aware of for interaction with the OSC Bridge and the Python toolset environment.

- The "**extra**/" folder: This is the folder which contains the "extra" (additional, goal specific) Python methods (scripts) divided in sub-folders per type. These scripts are both used for automation and by the AI Tool for handling actions. Consider this folder and its sub-folders as "read-only". Do not delete files and (unless you are brave) modify them (if you do, I will not be able to support your changes).

- The "**help**/" folder: This folder contains Gemini CLI additional information for the "Help: <keyword>" function, useful for inline help on prompt examples. Consider this folder "read-only" too.

- The "**init**/" folder: This folder contains Gemini CLI additional information pointed by the "osc-bridge-guidelines.md" file. Consider this folder "read-only". Do not delete its content.

- The "**tmp**/" folder: This folder starts empty and it is a folder where Gemini CLI is allowed to create new code (Python scripts) for actions it cannot fulfill with methods in the "core/" and "extra/" folder. You are allowed to clean up this folder if you wish, although these files are miniscule, so do not worry too much about it growing too much.

The (JSON) Data Formats

Let's look at the data formats used by this environment. Either using the Python toolset for automation tasks or the interface for a "local-coding-enabled" AI Tool like Gemini CLI, you will need to be aware of the data formats used by the environment. These formats have been defined to be as portable as possible, matching as much as possible the internal data structures in Ableton Live.

The following data formats are defined:

JSON MIDI dictionary

This JSON dictionary defines a MIDI clip notes content, as a list of "notes" defined by the following tuple:

```
{
    "note_id": 0,
    "pitch": 69,
    "start_time": 0.0,
    "duration": 4.0,
    "velocity": 80.0,
    "mute": 0,
    "probability": 1.0,
    "velocity_deviation": 0.0,
    "release_velocity": 64.0
}
```

This JSON format is formally defined in the "definition_midi_dict.json" file in the defs/ folder.

Side-Note: the same JSON MIDI dictionary format is used by the Max for Live "J74 MIDI Assistant" plug-in. This means that MIDI data created or manipulated with the "J74 Assistant Developers Tools" package (either via the Python toolset for direct automation or by the AI Tool used with the package) can be *interchanged* with data created or manipulated with the "J74 MIDI Assistant" plug-in.

JSON Warp Markers dictionary

This JSON dictionary defines an audio clip warp markers content. It contains a list of "warp markers" defined by the following tuple:

```
{
    "beat_time": 1.0093879699707031,
    "sample_time": 0.5046939849853516
}
```

This JSON format is formally defined in the "definition_warp_dict.json" file in the defs/ folder.

JSON Device Snapshot dictionary

This JSON dictionary defines a snapshot for a device in a track. It contains a list of all the parameters of the device (as tuples, with among others the actual value and the value range), plus additional metadata (e.g. the device type):


```
{
    "parameters": [
        {
            "index": 0,
            "name": "Device On",
            "value": 1.0,
            "min": 0.0,
            "max": 1.0
        },
        {
            "index": 1,
            "name": "GainLo",
            "value": 0.7647867202758789,
            "min": 0.0,
            "max": 1.0
        },
        …
        ,
        {
            "index": 9,
            "name": "Slope",
            "value": 0.0,
            "min": 0.0,
            "max": 1.0
        }
    ],
    "metadata": {
        "class_display_name": "EQ Three",
        "track_number": 2,
        "device_number": 2,
        "timestamp": "2026-04-07 15:56:02"
    }
}
```


This JSON format is formally defined in the “definition_device_snapshot.json” file in the defs/ folder.

JSON Mixer Snapshot dictionary

This JSON dictionary defines a snapshot for the Mixer. It contains a list of all the collected tracks and their parameters (as tuples), plus additional metadata (e.g. the Mixer number and type of tracks):


```
{
    "tracks": [
        {
            "name": "1-MIDI",
            "volume": 0.8500000238418579,
            "panning": 0.0,
            "sends": [
                0.0
            ],
            "mute": false,
```

```
            "solo": false,
            "arm": true
        },
        ….
        ,
        {
            "name": "6-Audio",
            "volume": 0.8500000238418579,
            "panning": 0.0,
            "sends": [
                0.0
            ],
            "mute": false,
            "solo": false,
            "arm": false
        }
    ],
    "return_tracks": [
        {
            "name": "A-A-A-Filter",
            "volume": 0.8500000238418579,
            "panning": 0.0,
            "sends": [
                0.0
            ],
            "mute": false,
            "solo": false,
            "arm": null
        }
    ],
    "master_track": {
        "name": "Master",
        "volume": 0.8500000238418579,
        "panning": 0.0,
        "sends": [],
        "mute": null,
        "solo": null,
        "arm": null
    },
    "metadata": {
        "track_count": 6,
        "return_track_count": 1,
        "check_value": 1006,
        "timestamp": "2026-04-11 13:22:50"
    }
}
```


This JSON format is formally defined in the “definition_mixer_snapshot.json” file in the defs/ folder.

Tip: you will collect and generate lots of JSON data file examples. Using the toolset in its standard form, all this JSON data will be collected and generated in the “data/” folder. IT will have a “.json” file extension.

Additional Notes on Gemini CLI

Here are a few additional notes and remarks about Gemini CLI. A full review of Gemini CLI is far beyond the scope of this project and would require a separate blog or document. Anyway, a few notes, remarks and links can be helpful. If you are not interested, you can skip this section.

About learning Gemini CLI (as a developer):

Here a few interesting links to understand more about Gemini CLI, as provided by Google:
https://geminicli.com/docs/reference/commands/
https://geminicli.com/docs/get-started/installation/
https://github.com/google-gemini/gemini-cli

Here a few blogs about Gemini CLI, as provided by experts in this field:
https://www.datacamp.com/tutorial/gemini-cli
https://addyosmani.com/blog/gemini-cli/#tip-13-multi-directory-workspace-one-gemini-many-folders
https://blog.openreplay.com/tips-tricks-gemini-cli/

About "quota" management:

Gemini CLI, in its unpaid free tier, gives users a generous "quota" (= amount of CPU capacity in tokens which you can consume from the Google AI data-centers). While generous (from Google), that quota is not infinite. Therefore, you will need to be aware of the usage you make, track it if possible. This is important because not only you need to know when you exceed that quota (as you will be temporary downgraded in AI model or even stopped for the rest of the day), but also to prevent you exceed the maximum quota, as every time you exceed the "back off" timer will be increased (that is if you exceed once you will be stopped for a few minutes, if you do it again in the same day you will stopped for a few hours etc. etc.). Hence: try to avoid exceeding the quota.

Within Gemini CLI there is one command which lets you check the quota you use:

```
> /stats
```

About the "md" instructions used for the J74 Assistant Developers Tools:

Strictly speaking you do not need to know the insides of the mechanisms in Gemini CLI to use this environment, but if you wish to modify it, develop it or maybe just understand it is worthy having a look and understand the "./docs/osc-bridge-guidelines.md" descriptor file.

Side note: Gemini CLI uses a hierarchy of md (markdown) descriptors for the logic needed to understand a given code repository. A Gemini CLI .md file is a plain-text document used to interact with Google's Gemini AI through a Command Line Interface (CLI).

At any time, you can ask Gemini CLI to show the content of its memory loaded by md files with the following:

```
> /memory show
```

While most people use Gemini through a browser, developers and power users often use CLI tools to send prompts. These tools frequently use .md files as a clean way to store long, complex prompts or to save the AI's responses in a format that is easy to read and share.

What does "md" stand for? MD stands for Markdown. It is a lightweight "markup language" created in 2004 by John Gruber. The goal was to allow people to write using an easy-to-read, easy-to-write plain text format that can be automatically converted into structurally valid HTML.

Below a copy of the content of the "./docs/osc-bridge-guidelines.md" file. This might be useful as a read and for explanation purposes (have a look, it is quite readable!):

```
# GEMINI.md - Ableton Live OSC Bridge Workflow
This file defines the foundational mandates and workflows for the J74 OSC Bridge environment. These
instructions take absolute precedence over general defaults.

## [1] Core Mission & Execution Protocol
You are an expert Ableton Live assistant using the LOM API via OSC.
1. **Trigger:** Respond to "Action: xxxx" by choosing the appropriate script.
2. **Lookup:** Map requests to Atomic Actions [2] or Composite Actions [3]. Use **[z]** (Raw LOM) if no direct
match exists.
3. **Reference:** Use `core/` for methods, `extra/` (and its sub folders) for additional examples, and official
LOM docs for deep logic.
4. **Constraints:**
   - Prefer Atomic Actions.
   - Save new scripts to `tmp/`.
   - Save data JSONs to `data/`.
   - **Unsupported:** Do not attempt to add/delete devices, write automation, edit Arrangement clips, or load
files/samples.

## [2] Atomic Action Mapping Table
| Code | Objective | Script Path |
| :--- | :--- | :--- |
| **[a]** | Extract MIDI notes to JSON | `core/collect_midi_clip.py` |
| **[b]** | Create MIDI clip from JSON | `core/create_midi_clip.py` |
| **[c]** | Extract warp markers to JSON | `core/collect_warp_markers.py` |
| **[d]** | Add warp markers from JSON | `core/create_warp_markers.py` |
| **[e]** | Shift selected warp marker | `core/move_warp_marker_relative_distance.py` |
| **[f]** | Capture device snapshot | `core/device_snapshot.py` |
| **[g]** | Restore/Verify device snapshot | `core/restore_device_snapshot.py` |
| **[h]** | Capture mixer snapshot | `core/mixer_snapshot.py` |
| **[i]** | Restore mixer snapshot | `core/restore_mixer_snapshot.py` |
| **[j]** | Clear warp markers (except ends) | `core/clear_warp_markers.py` |
| **[w]** | Test OSC Loopback | `core/verify_loopback.py` |
| **[z]** | Raw LOM Action (call, set, get) | Script: `core/lom_command_osc.py`. Usage: `python
core/lom_command_osc.py "<path>" <action_type> <property> [<value>]`. Use for direct LOM interaction without a
dedicated script. |

## [3] Composite Action Mapping Table
| Code | Objective | Logic |
| :--- | :--- | :--- |
| **[composite_1]** | Modify MIDI Clip | 1. Collect ([a]) -> 2. Modify JSON -> 3. Push ([b]). |
| **[composite_2]** | Time-stretch Audio Clip | 1. Collect ([c]) -> 2. Modify `beat_time` -> 3. Clear ([j]) ->
4. Push ([d]). |
| **[composite_3]** | Replace Warp Markers | 1. Collect ([c]) -> 2. Add/Remove in JSON -> 3. Clear ([j]) -> 4.
Push ([d]). |

## [4] Environment Configuration (`core/settings_osc_bridge.py`)
- **Ports:** Send: 4470 | Recv: 4471
- **Delays:** MIDI/Warp: 0.2s | Standard: 0.05s | Special: 0.01s
- **Collection Time:** 2.5s for MIDI/Warp data gathering.

## [5] Directory Structure
- **`core/`, `extra/`, {sub folders of `extra/`}:** Protected (Read-Only).
- **`defs/`:** Location of formal definitions and examples of JSON objects for data storage
- **`help/`:** Location for help files. Protected (Read-Only).
```

```
- **`init/`:** Location for init Instructions.txt file. Protected (Read-Only).
- **`data/`:** Work folder for data JSON files.
- **`tmp/`:** Work folder for new/modified scripts.

## [6] Indexing and JSON Format
- **User Inputs:** Treat numbers as 1-based for Atomic Scripts ([a-j]). The scripts handle the conversion.
- **Raw LOM ([z]):** ALWAYS use 0-based indices.
- **Absolute Notation:** If user says "clip 2 0", treat as 0-based absolute indices.
- **JSON Formats:** Uses the JSON object definitions for data as in `defs/`

## [7] Help Protocol
- **Help:** If "Help: yyyyy" is triggered, find the match in `help/Help_CLI.txt`.
   - Start with: `# Examples of action prompts for the yyyyy category:`
   - Print content between `<help content: yyyyy>` and `</help content: yyyyy>`.

## [8] Raw LOM [z] Execution Strategy
When performing Raw LOM actions (Code **[z]**), choose the most efficient method based on task complexity:
1. **Single Action:** Use `python core/lom_command_osc.py` with inline parameters.
2. **Short Sequence (2-5 actions):** Use multiple sequential calls to `python core/lom_command_osc.py`.
3. **Long Sequence or Bulk Operations (e.g., "create 64 tracks"):** Create a specialized ad-hoc script in
`tmp/`. This is more efficient as it reuses a single OSC client/server instance and can utilize Python loops
for repetitive tasks.
```

Comments:

- [1] "*Core Mission & Execution Protocol*": this defines the main rules of engagement. The user is supposed to give a *prompt* for Ableton Live tasks using the "**Action:**" prompt structure. The AI will then parse the request and identify the action using look-up tables [2] and [3].

- [2] "*Atomic Action Mapping Table*": this is a table which points to the Python "core" methods, methods which perform atomic operations on Ableton Live.

- [3] "*Composite Action Mapping Table*": this is a table which points to the Python "core" and "extra" actions, actions which require the concatenation of multiple atomic methods.

- [4] "*Environment Configuration*": the definition of start-up options and parameters. These parameters are editable, as they are saved in the "core/settings_osc_bridge.py". The recommendation is NOT to change these parameters, as they should be working well for most environments.

- [5] "*Directory Structure*": this tells the AI Tool where things are.

- [6] "*Indexing and JSON Format*": instructions for the AI on the 0-based indexing used by the LOM API of Ableton Live and on the use and definition of JSON formats for data.

- [7] "*Help Protocol*": defines the inline help protocol. You can ask for example prompts using the "Help: <keyword, like MIDI or Track or Device>" prompt structure.

- [8] "*Raw LOM [z] Execution Strategy*": this gives the AI a strategy for deciding whether to run atomic calls or create (potentially far more efficient) ad-hoc scripts.

Limitations

Operations that the "J74 Assistant Developers Tools" can perform, and an AI (such as Gemini CLI) can perform as companion assistant, are limited by the extent of the Ableton Live LOM API implementation. That is: operations not possible through Ableton Live LOM API, will also not be possible for the toolset and for AI tool working as a companion assistant.

A list of what is supported (and documented) by Ableton on the LOM API is here:

https://docs.cycling74.com/legacy/max8/vignettes/live_object_model

Here is a non-exhaustive summary of what is currently possible *[in square brackets how this package handles them]*:

Transparent operations (scripts can use API calls directly)

Track operations [transparent]

- create
- delete
- duplicate
- color
- rename

Track Mixer operations [transparent]

- mute
- volume
- panning
- send
- arm
- solo

Clip (slot) operation [transparent]

- create
- delete
- duplicate
- start
- stop
- loop
- color

Application State operations [transparent]

- get version
- get track number / overview
- undo

Device Parameter operation [transparent]

- read value (and range)
- set value

Guided operations (the toolset handles Ableton Live internal data sets)

MIDI clip note operations [guided]

- creation of MIDI clips with notes
- modification of MIDI clips with notes

Audio clip warp operations [guided]

- add warp markers
- remove warp markers
- move warp markers

Scripted operations (the toolset builds data proprietary sets)

Device Snapshot operation [scripted]

- collect all device parameter values (and ranges) into a device snapshot (JSON file)
- restore all device parameter values from a device snapshot
- interpolate device parameter values across device snapshots
- randomize device parameter values within ranges or constraints

Mixer Snapshot operation [scripted]

- collect all mixer parameter values (and ranges) into a mixer snapshot (JSON file)
- restore all mixer parameter values from a mixer snapshot
- interpolate mixer parameter values across mixer snapshots
- randomize mixer parameter values within ranges or constraints

Operation NOT possible (as not possible in Ableton Live LOM API)

Here is a non-exhaustive summary of what is (currently) NOT possible:

- Any activity which involves the Ableton Live “browser”, such as adding or deleting devices in the live set, loading a sound or a sample, loading a preset, loading a plug-in and so on.
- Working on automation on any type of clip (arrangement or session view)
- Working on arrangement view clips (the possible operations only target session view clips)

Licensing, Liability, Support

The purchase of a license for the “J74 Assistant Developers Tools“ package gives rights for the use of the package content (the Max for Live device, the python scripts, the AI environment, all parts of the package).

The purchase of the J74 Assistant package does NOT include rights on the use of the external AI Tool: this must be separately provided by the user.

Liability: The user (producer) always remains responsible for the AI tool actions, which potentially can be disruptive if left unchecked. Therefore: *save often your project when using an external AI*.

No liability whatsoever goes to the maker of these tools.

In case support issues, please report those via email to: info [at] fabriziopoce.com

Appendix [1] Installation and Setup #*[Detailed Version]*

This appendix provides additional details, comments and explanation for the installation and setup of the "J74 Assistant Developers Tools". Please follow these steps carefully. Try to understand what each step does and requires. Move on to the next step only if you are sure the previous step was completed successfully.

[Step-1] – Download and Unzip the ZIP archive

To start *download* the package archive from the link you received after the purchase. Once the download has completed, *unzip* the download pack. Once unzipped, you will see the following content:

- The *OSC Bridge*, a Max for Live device plug-in named: "Assistant_Bridge_J74.amxd"
- The *J74 Assistant code repository*, a folder named "J74_Assistant_Code"
- A set of *PDF documents*, including this user manual.

[Step-2] – Placement of the Max for Live "OSC Bridge" plug-in in the Ableton Live project

The OSC Bridge ("Assistant_Bridge_J74.amxd"), is a transparent Max for Live audio device. It needs to be present in the Ableton Live project for the use cases addressed by the "J74 Assistant Developers Tools" to work. The preference is to have it running on the Master track, as this track cannot be deleted, but any other track would also do. The plug-in is completely transparent to audio and does not process or affect audio passing through.

You can save (copy) the OSC Bridge ("Assistant_Bridge_J74.amxd") anywhere on your system, in any location you like. When you need it in Ableton Live, just drag and drop it from the Ableton Live browser onto the selected track. You can load it and leave it there, as you do not need to configure anything on it (its default will do).

Tip: you can include the location of the folder where the "Assistant_Bridge_J74.amxd" is located directly in the Ableton Live browser, go to the "Places" section of the Ableton Live browser and select [+] Add Folder...

[Step-3] – Placement of the "J74 Assistant Code" repository (folder) on your system

The J74 Assistant code repository is the folder named "J74_Assistant_Code" which you unpacked in the previous steps. Decide for a place on your computer drives where you wish to keep it. Copy it there. You can do it with the usual "File Explorer" (on Windows) / "Finder" (on a MAC) copy & paste actions or use the OS copy utilities below.

On a Windows system you can use the utility "robocopy" for copying all its content to the destination place:

```
robocopy "[SOURCE_PATH]\J74_Assistant_Code" "[DESTINATION_PATH]\J74_Assistant_Code" /E /Z /MT:16
```

Example:

```
robocopy "C:\Users\Fabbri\Downloads\J74_Assistant_Code" "D:\My Projects\J74 Assistant\J74_Assistant_Code"
```

On a MAC OS system you can use the utility "rsync" for copying all its content to the destination place:

```
rsync -avzhP --inplace "[SOURCE_PATH]/J74_Assistant_Code" "[DESTINATION_PATH]/J74_Assistant_Code"
```

Example:

```
rsync -avzhP --inplace "/Users/j74_assistant/Downloads/J74_Assistant_Code"
"/Users/j74_assistant/Documents/J74_Assistant_Code"
```

IMPORTANT: please note down the location of the folder assigned to the code repository. *You will need that later.*

[Step-4] – Python Installation

Next, we will look at the Python environment. The code repository consists of a set of Python scripts and classes made to work with the J74 OSC Bridge plug-in, plus additional configuration items for the AI Tool (Gemini CLI). This toolset, combined with the J74 OSC Bridge, creates an MCP (Model Context Protocol) server for AI applications while also functioning as an automation toolset by itself.

Before you can start using it, we need to look at the necessary dependencies, first the presence of a *Python Interpreter*. The code toolset in fact requires the presence of a Python Interpreter on the OS to be used.

MAC OS: On the MAC system Python is already per-installed with the system, but it can be an older version (you are advice to use a Python version which is 3.11 or newer). Therefore, after checking the pre-installed version (see next) we might need to install a newer version from https://www.python.org/

Windows: On a Windows system it is an add-on feature anyway, so we will typically have (unless you installed python before) to install it using the installer from https://www.python.org/.

*Tip: you can check if the python interpreter is already present on your system. Open a shell (command prompt) and type "*python --version*". If python is installed, you should see the version running (example "Python 3.13.13")*

ACTION: If you need to install python on your system, you should follow the instructions from https://www.python.org/. Here is a summary of the installation process:

- Download the Installer from the official Python. For most users, the 64-bit installer is the correct choice.
- Start the installer. If asked, make sure that the box "Add Python to PATH" is activated. This allows Python scripts to run from any location in the Shell terminal. Conclude the Installation process.

Other remarks users:

- MAC OS: If you keep the OS version of python, run all the scripts using the "python" inline. If you instead decide to install the newer version, use the "python3" inline command instead of "python.
- Windows OS: After installing Python run all the scripts using the "python" command

[Step-5] – The OSC library for Python ("python-osc")

Once you assured the Python interpreter is installed, you need to verify that the *"python-osc" library* is also present in Python. This library is a non-default library, and it is needed to let Python scripts "talk" OSC. You can check and eventually add the library directly from the Shell (Command Prompt).

On Windows and, if you use the system Python interpreter, on MAC OS use the "pip" utility of Python, in this way:

```
$$ pip install python-osc
 Downloading python_osc-1.9.3-py3-none-any.whl.metadata (6.4 kB)
 Downloading python_osc-1.9.3-py3-none-any.whl (43 kB)
 Installing collected packages: python-osc
 Successfully installed python-osc-1.9.3
```

Or, if you are on a MAC OS system but you installed the newest version, do:

```
$$ pip3 install python-osc
 Downloading python_osc-1.9.3-py3-none-any.whl.metadata (6.4 kB)
 Downloading python_osc-1.9.3-py3-none-any.whl (43 kB)
 Installing collected packages: python-osc
 Successfully installed python-osc-1.9.3
```

This completes the set-up for the Python part. Next, we will look at the set-up of the AI Tool, Gemini CLI.

[Step-6] – Install the "local-coding-enabled" Gemini CLI AI Tool

The following summarizes the major steps in getting Gemini CLI, the target AI Tool, to run on your computer.

Disclaimers:

- Gemini CLI is an AI Tool from *Google*. The "J74 Assistant Developers Tools" and its maker has no direct affiliation whatsoever with Google or any other AI platform. Gemini CLI has been chosen as a reference AI Tool due to its *generous free tier* and of course *for its "local-coding-enabled" capabilities*.
- To run Gemini CLI a (free) Google account (as with GMAIL) is required. Be sure you have a Google account before starting this step. When required (e.g. at the first launch) you will be asked to log in to that account to get the Gemini CLI AI Tool to run. Please make sure the log-in works.
- An AI Tool like Gemini CLI does NOT run entirely locally on your computer. It has an interface to your computer locally (for the prompts, for the code and for its file usage) but it also talks to the Google back-end computing data-centers via the Internet, where all its "intelligence" (AI models, neural networks and data) resides. Therefore, you MUST have your computer connected to the Internet to be able to use Gemini CLI.

Before starting you need to be aware of the following:

- *Gemini CLI runs on top of Node.js*, so this means that you will need to install Node.js first.
- Node.js packages and instructions can be found here: https://nodejs.org/en/download
- Gemini CLI instructions can be found here: https://geminicli.com/docs/get-started/installation/

IMPORTANT: Node.js is OS dependent and requires you to follow instructions carefully, as explained next.

- On Windows, Node.js supports the same installers on both Windows 10 and Windows 11. As far as I have seen, installation with the standard Node.js installer worked well on all the Windows systems I tested

- On MAC OS it depends on the OS version you run. At the time of writing, the following applies:
  - *On MAC OS 14 (Sonoma) and newer*, the generic installer (.pkg) provided by the Node.js site for the latest Node.js version 24 should work. Please use that installer as your primary target. Only if that fails fall back to the procedure described in Appendix [2] of this user manual
  - *On older versions of MAC OS, from MAC OS 13 (Ventura) down to MAC OS 11 (Big Sur)*, you need to install an older version of Node.js (such as version 22, a version which still satisfies Google CLI requirements). In this case perform the procedure described in Appendix [2] of this user manual.

[Node.js Installation]

ACTION: As of the time of writing, refer to the latest instructions from Node.js here:
https://nodejs.org/en/download

VERIFICATION: to verify Node.js is running, after the installation of Node.js, do:
node -v # This should print the version of Node.js like "v24.15.0" or "v22.22.2" or whatever is the version you installed. # If you do not get the version number, but some error, please refer to Appendix[2] to fix the Node.js installation.

[Gemini CLI Installation]

After Node.js is installed, follow the instructions provided by the Gemini CLI site:

ACTION: As of the time of writing, refer to the latest instructions from Google here:
https://geminicli.com/docs/get-started/installation/

DETAIL: as shown on the link above, the install of Gemini CLI is a library install in Node.js, done with this command:
npm install -g @google/gemini-cli

VERIFICATION: we will verify Gemini CLI is running in [Step-8] next. We need first to go through [Step-7]

Again, follow these instructions carefully and control the requirements before installation.

[Step-7] – Start Gemini CLI (log-in, trust of the code repository folder)

Once the installation process for Gemini CLI (and before that, Node.js) has been completed, *navigate to the folder you assigned to the code repository* (the location you have chosen in [Step-3], where you copied the code repository) and type "***gemini***".

```
D:\>cd "D:\My Projects\J74 Assistant\J74_Assistant_Code"
D:\My Projects\J74 Assistant\J74_Assistant_Code> gemini
```

The first time you do this Gemini CLI will require you to:

- *Trust of the selected "J74_Assistant_Code" code repository folder*
- *Authenticate with your Google account*

You need to do both.

This is what you should get:

```
************************************************************************************************
* Do you trust the files in this folder?
*
* Trusting a folder allows Gemini CLI to load its local configurations, including custom commands, hooks, MCP
* servers, agent skills, and settings. These configurations could execute code on your behalf
*
*   [*] 1. Trust folder (J74_Assistant_Code)
*       2. Trust parent folder (J74 Assistant)
*       3. Don't trust
*
************************************************************************************************
```

IMPORTANT: You need tp select "1. Trust folder" in order to use the tools of this package.

After logging in and trusting the folder, you should see something like this:

```
Gemini CLI is restarting to apply the trust changes...

 ***      Gemini CLI v0.38.2
   ***
  ***     Signed in with Google /auth
 **       Plan: Gemini Code Assist for individuals /upgrade

Logged in with Google: xxxxxxxx@gmail.com /auth
```

At this point, in theory, all should be working properly. In the next steps we will verify that indeed all is working.

[Step-8] – Verifications for Gemini CLI and for the environment

You can quickly verify that the set-up for Gemini CLI has worked by doing the following in Gemini CLI:

> /memory show

```
ℹ Current memory content from 1 file(s):
  ---
  --- Project ---
  --- Context from: g:/J74_Assistant_Code/gemini.md ---
  <!-- Imported from: ./docs/osc-bridge-guidelines.md -->
  # GEMINI.md - Ableton Live OSC Bridge Workflow

This file defines the foundational mandates and workflows for the J74 OSC Bridge environment. These
instructions take absolute precedence over general defaults.
....
```

This Gemini CLI command shows the content of any descriptive file pre-loaded in memory and, if all worked as expected, it should show the content of the "./docs/osc-bridge-guidelines.md" a part of the code repository of the "J74 Assistant Developers Tools", which Gemini CLI should have loaded when started in the trusted folder of the code repository.

If that (see above snippet) is what you get, you can now proceed performing a *connectivity test* to the OSC Bridge in Ableton (of course, please make sure the OSC Bridge is running in Ableton Live *before* you start this test).

From the command line of Gemini CLI issue this *prompt* (you do not need to copy the leading ">"):

> Action: verify connectivity with Ableton Live

If the Gemini CLI AI Tool has loaded the environment properly, it will understand and automatically run a test Python script which tests both Max for Live and Python assets. If that is working, this will not only ensure connectivity but also the proper recognition and functioning of the entire Python environment in the AI Tool.

IMPORTANT: Gemini CLI will sometimes require permission to write and read files and for running code. This is normal. When asked, grant the permission using the "**Allow for this session**" option. In theory you can avoid this using the Gemini CLI "YOLO" ("you only live once") option, but that option is not recommended (for your computer safety) as it might allow Gemini CLI too much freedom on your computer. My recommendation is never to use the YOLO option in Gemini CLI (trust me, it did cause me headaches...).

If that prompt worked, you should have something like the following result:

```
> Action: verify connectivity with Ableton Live

✦ I will run the loopback test to verify the OSC connectivity with Ableton Live.
╭──────────────────────────────────────────────────────────────────────────────╮
│ ✓  Shell Running OSC loopback test to verify connectivity with Ableton Live. │
│                                                                              │
│ ... first 1 line hidden (Ctrl+O to show) ...                                 │
│                                                                              │
│ [TEST] Sending loopback test message: /loopback_down                         │
│                                                                              │
│ [OSC SEND] Message: /loopback_down                                           │
│ Waiting up to 3.0s for /loopback_up ACK...                                   │
│                                                                              │
│ [OSC RECV] Data: /loopback_up ACK                                            │
│                                                                              │
│ [SUCCESS]: Received ACK from bridge. Connection is working.                  │
╰──────────────────────────────────────────────────────────────────────────────╯
✦ Connectivity with Ableton Live has been verified; the OSC loopback test was successful.
```

You can also verify that the “Help” prompt function for the environment is working (you can use the simple with a prompt “Help:” <followed by some keyword>):

```
> Help: MIDI

✦ I will read the MIDI help content from the CLI help file.
  ✓  ReadFile  help\Help_CLI.txt

✦ Examples of action prompts for the MIDI category:

  > Action: create a midi clip in track 1 slot 1 with a 4 bar long C minor chord
  > Action: create a midi clip in track 1 slot 1 with a chord progression IV V I VI in the G minor scale
  > Action: create a midi clip in track 1 slot 1 with a progression I V IV II in C major 1 bar per chord
  [etc. etc.]
```

Next as a simple API/OSC test, ask the AI to mute / unmute a track in the current live set, with a *prompt* like this:

```
> Action: mute track 1
```

And then:

```
> Action: unmute track 1
```

This test will show you how the AI can run LOM API commands. It should look like below:

```
> Action: mute track 1
*************************************************************************************************************
I will use the Raw LOM Action ([z]) to mute track 1 by setting its mute property to 1. Since this is a single
atomic action, I will call core/lom_command_osc.py directly with inline parameters.
*************************************************************************************************************
* Shell python core/lom_command_osc.py "live_set tracks 0" set mute 1
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* Sending LOM Command:
*   Path:        live_set tracks 0
*   Action:      set
```

```
*   Property:    mute
*   Value:       1
* [INFO] Command sent. No specific return value received (common for 'set' actions).
* Track 1 has been muted. (LOM: live_set tracks 0, set mute 1).

> Action: unmute track 1
************************************************************************************************************
I will unmute track 1 by setting its mute property to 0 using the Raw LOM Action ([z]) via
core/lom_command_osc.py.
************************************************************************************************************
* Shell python core/lom_command_osc.py "live_set tracks 0" set mute 0
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* Sending LOM Command:
*   Path:        live_set tracks 0
*   Action:      set
*   Property:    mute
*   Value:       0
* [INFO] Command sent. No specific return value received (common for 'set' actions).
* Track 1 has been unmuted. (LOM: live_set tracks 0, set mute 0).
```

Finally, as a more advanced test, ask the AI Tool to build a MIDI clip (in this example track 1 is a MIDI or instrument track – if that is not the case, please change the prompt below accordingly). The *prompt* should be something like:

> Action: create a 1 5 6 4 MIDI progression in the C Major key in track 1 slot 1

The effect should be something like this:

```
> Action: create a 1 5 6 4 MIDI progression in the C Major key in track 1 slot 6
************************************************************************************************************
✦ I will create a MIDI progression (I-V-vi-IV) in C Major (C Maj, G Maj, A min, F Maj) for track 1, slot 6.

Following the execution strategy, I will first use the Raw LOM script([z]) to create a new MIDI clip at the
target location, then use the composite action logic ([composite_1]) to populate it with the progression.

  Step 1: Create the MIDI clip (4 bars)
  I'll use core/lom_command_osc.py to create the clip.
  LOM: live_set tracks 0 clips 5 (1-based user track 1, slot 6).
  Action: call create_clip 16.0 (4 bars = 16 beats).

  Step 2: Define and Push MIDI Data
  I'll create a temporary JSON file with the notes and then use core/create_midi_clip.py ([b]) to push them.

  Chords (C Major):
   1. C Maj (I): [C3, E3, G3] -> [48, 52, 55] | Start: 0.0, End: 4.0
   2. G Maj (V): [G2, B2, D3] -> [43, 47, 50] | Start: 4.0, End: 8.0
   3. A min (vi): [A2, C3, E3] -> [45, 48, 52] | Start: 8.0, End: 12.0
   4. F Maj (IV): [F2, A2, C3] -> [41, 45, 48] | Start: 12.0, End: 16.0

  Starting the process now.
************************************************************************************************************
* Shell python core/lom_command_osc.py "live_set tracks 0" call create_clip 16.0
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* Sending LOM Command:
*   Path:        live_set tracks 0
*   Action:      call
*   Property:    create_clip *   Value:       16.0
* [INFO] Command sent. No specific return value received (common for 'call' actions).
************************************************************************************************************
 ✓  WriteFile  C_Major_1564_progression.json → Accepted (+16, -0)

      1 {
      2   "notes": [
      3     {"pitch": 48, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      4     {"pitch": 52, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      5     {"pitch": 55, "start_time": 0.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      6     {"pitch": 43, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
```

```
       7     {"pitch": 47, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       8     {"pitch": 50, "start_time": 4.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
       9     {"pitch": 45, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      10     {"pitch": 48, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      11     {"pitch": 52, "start_time": 8.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      12     {"pitch": 41, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      13     {"pitch": 45, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0},
      14     {"pitch": 48, "start_time": 12.0, "duration": 4.0, "velocity": 90, "probability": 1.0}
      15   ]
      16 }
************************************************************************************************************
✦ I will now use core/create_midi_clip.py ([b]) to push the C Major 1-5-6-4 progression notes into track 1,
slot 6. I'll use 1-based indexing as required by the atomic script.
************************************************************************************************************
* Shell python core/create_midi_clip.py 1 6 data/C_Major_1564_progression.json
* [INFO] 'clip_length' not found in JSON. Calculated as 16.0
* OSC Client initialized. Sending to 127.0.0.1:4470, Listening on 127.0.0.1:4471
* [INFO] Creating new clip with length 16.0...
* Adding 12 notes to Track 1, Slot 6...
* Successfully created MIDI clip on track 1 clip 6 from LOM format JSON.
************************************************************************************************************
✦ The 1-5-6-4 MIDI progression (C Maj, G Maj, A min, F Maj) has been created in Track 1, Slot 6.
   1. Clip Initialization: 4-bar empty clip created.
   2. Data Generation: Progression notes saved to data/C_Major_1564_progression.json.
   3. Execution: Notes pushed to the clip via the OSC Bridge.
```

If all that is working for you have done all the setup correctly!!!

From now on you can effectively use the package to its full extent and work with your AI Tool assistant (Google CLI) on any Ableton Live project. Please read the next paragraphs to learn how to use the toolset and how to make efficient prompts.

In case of set-up and installation issues, please report those via email to: info [at] fabriziopoce.com

Appendix [2] Older (or custom) Mac OS installation of Node.js

This procedure may be needed to complete [Step-6] of the installation process on a Mac OS system if [a] you have a Mac OS version older than macOS 14 Sonoma and/or [b] if the default installer from the Node.js site failed to install.

Tip: if you have MAC OS 14 (Sonoma) or newer, the generic installer (.pkg) provided by the Node.js site should work, so you should not need this procedure. Only if that does not work, use the procedure described next.
Remarks: On older versions of Mac OS, from Mac OS 13 (Ventura) down to Mac OS 11 (Big Sur), you will need to follow this procedure to install an older version of Node.js (version 22.x, which satisfies Google CLI requirements).
Remark: No support is possible from the J74 side on Mac OS versions older than Mac OS 11 (Big Sur).

[Option-1]: Install a previous version of Node.js using an installer (.pkg)

This option relies on installers made by the Node.js project for the previous LTS version 22.x, compiled on MAC OS versions older than MAC OS 14 (Sonoma), through previous versions of the Xcode Apple compiler.

If the default installer for the latest version of Node.js (at this time 24.x.x) did not work for you, try this option first; it is safe and quick (and version 22.x still comfortably matches the requirements from Google for Gemini CLI).

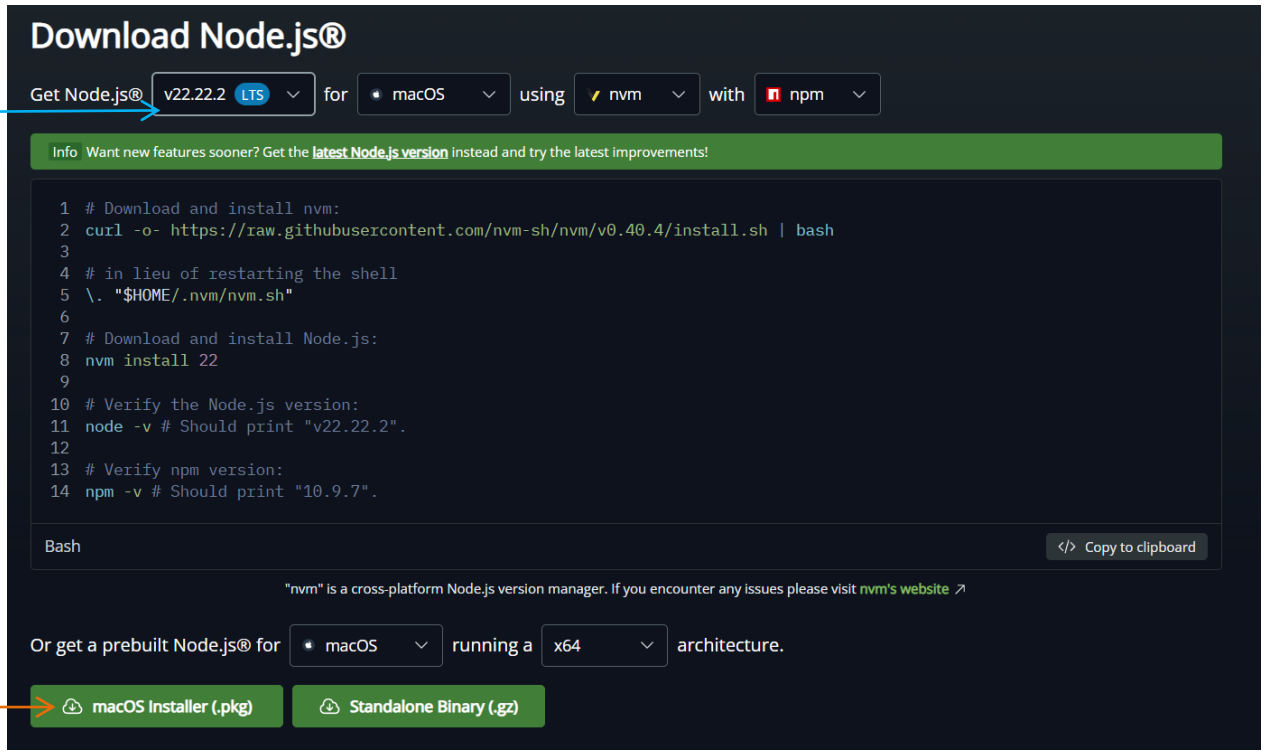
If this works (we will see later how to understand that), no source-code compilation on your Mac will be needed.

If this does not work you will need to proceed to option [2].

We start from the Node.js site (https://nodejs.org/en/download). This is what you get by default from the site:

At this point, if you select the “macOS Installer (.pkg)” it will get you the latest one, v24.15.0, which requires Mac OS 14 (Sonoma). Because we know we have an older Mac OS (older than Mac OS 14 – Sonoma) we change the version on the “Get Node.js®” (see *blue/cyan* arrow) to the older 22.x.x train (currently a selectable option is 22.22.2).

Like shown below:

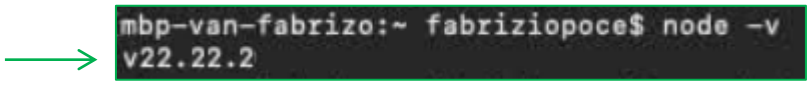


Now that we changed it to 22.22.2 we can hit the "macOS Installer (.pkg)". This will download the selected previous version. It will be something like "node-v22.22.2.pkg". Use that as your installer. Proceed with the installation.

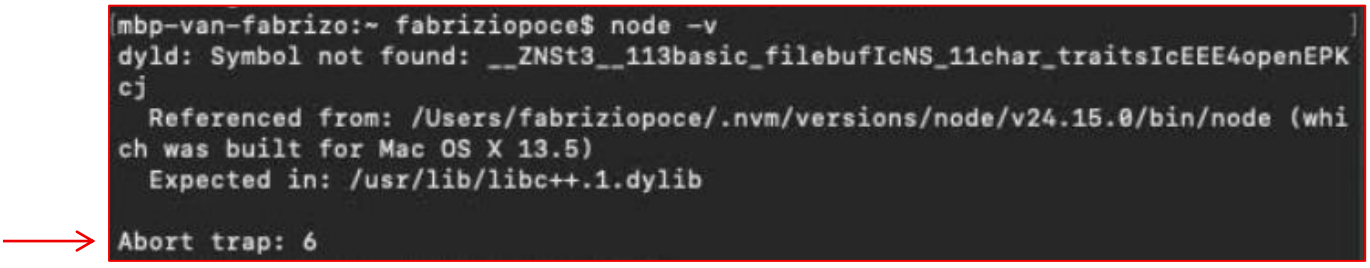
Checkpoint: Did the installation of Node.js work?

You can verify that by using this command after the installer finished the installation: "***node -v***"

If it did work, you should see a *version number* as a reply, a sign that Node.js is installed and can run. Like this:



If for some reason that did not work, you should see something (a bit "cryptic") like this:



The above is a *traceback error* (it says the package failed to run – it even says that the compiler was on a macOS 13.5). There is no version of Node.js. This installer did not work. <u>You need to perform option [2] described next.</u>

[Option-2]: Compile the source-code from Node.js in Xcode

This option will not rely on a pre-made installer but will use Xcode (from Apple) to compile the source code locally directly on your machine. In theory you could try it on any version of Node.js, but for compatibility, at this moment, you are recommended to do it on version 22.x.

The instructions are actually given on https://nodejs.org/en/download (see also previous pictures):

Download and install nvm (+ install Xcode if not present on your MAC):
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash
If necessary, install Xcode (if that is the case, then repeat the above “curl” command afterwards):
xcode-select --install
in lieu of restarting the shell
\. "$HOME/.nvm/nvm.sh"
Download and install Node.js:
nvm install 22
Verify the Node.js version:
node -v # Should print "v22.22.2".
Verify npm version:
npm -v # Should print "10.9.7".

Here is the actual process done on an older Mac OS system (in this example Mac OS 11, Big Sur).

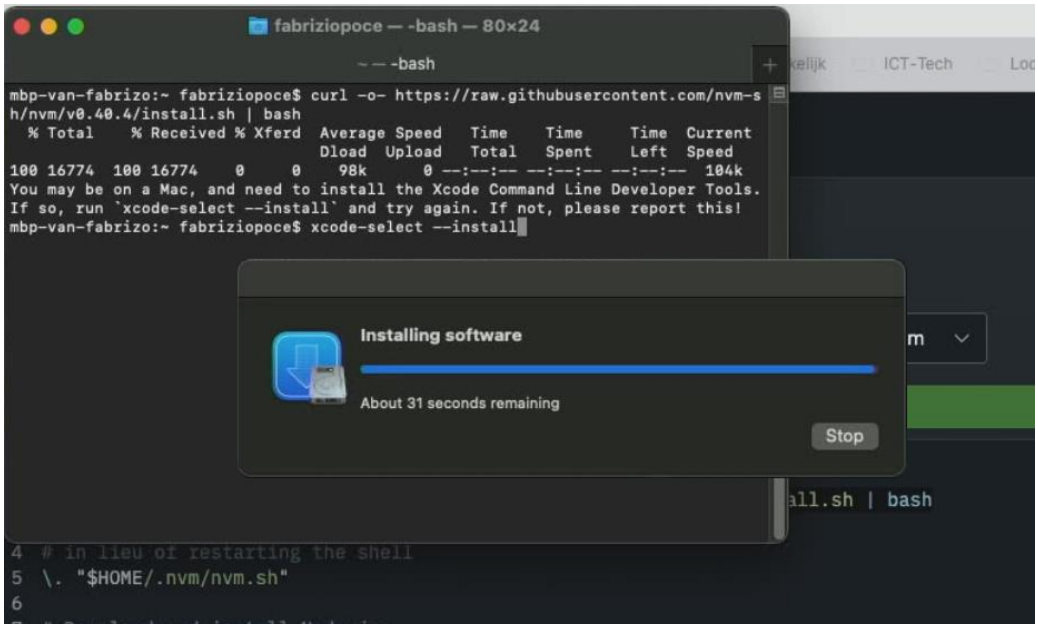
Download and install nvm (+ install Xcode if not present on your MAC):

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash

```
mbp-van-fabrizo:~ fabriziopoce$ curl -o- https://raw.githubusercontent.com/nvm-s
h/nvm/v0.40.4/install.sh | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 16774  100 16774    0     0    98k      0 --:--:-- --:--:-- --:--:--  104k
You may be on a Mac, and need to install the Xcode Command Line Developer Tools.
If so, run `xcode-select --install` and try again. If not, please report this!
```

Indeed, we need to install Xcode (it was not installed before on this Mac), by doing:

xcode-select --install

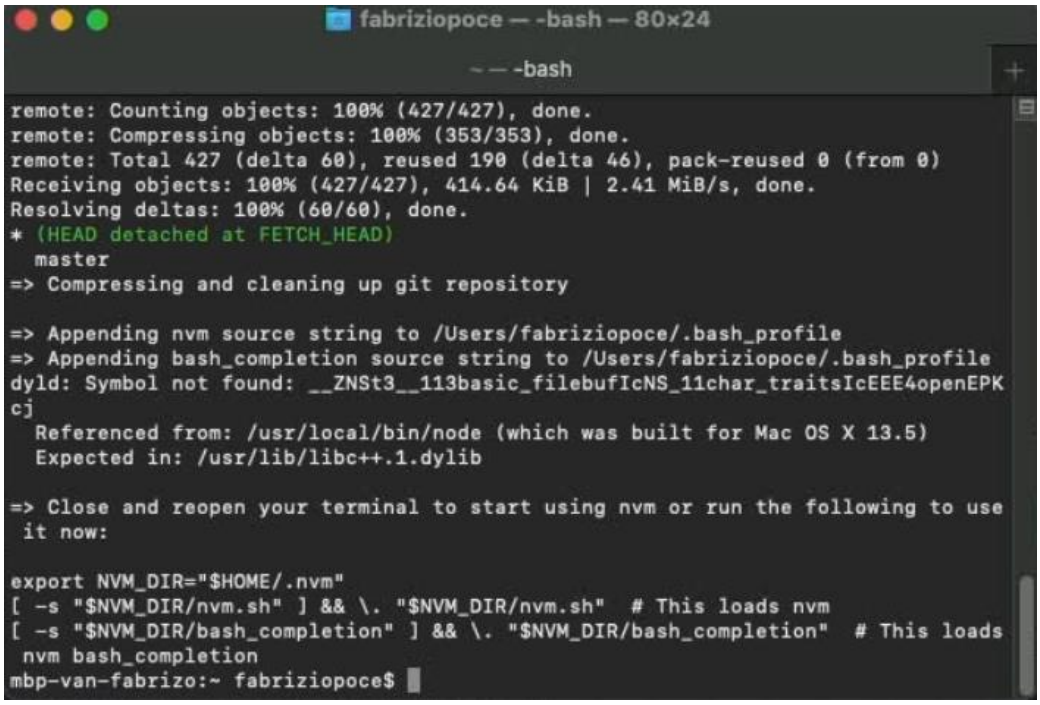


Remark: Apple's Xcode is big, so it might take a few minutes to download and install. Wait for it to be completed.

As the installation of Xcode was needed, after Xcode completion, we need to *repeat* the curl command:

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash

This time the output will be something like this:



Once you have successfully completed the curl command, restart the shell:

\. "$HOME/.nvm/nvm.sh"

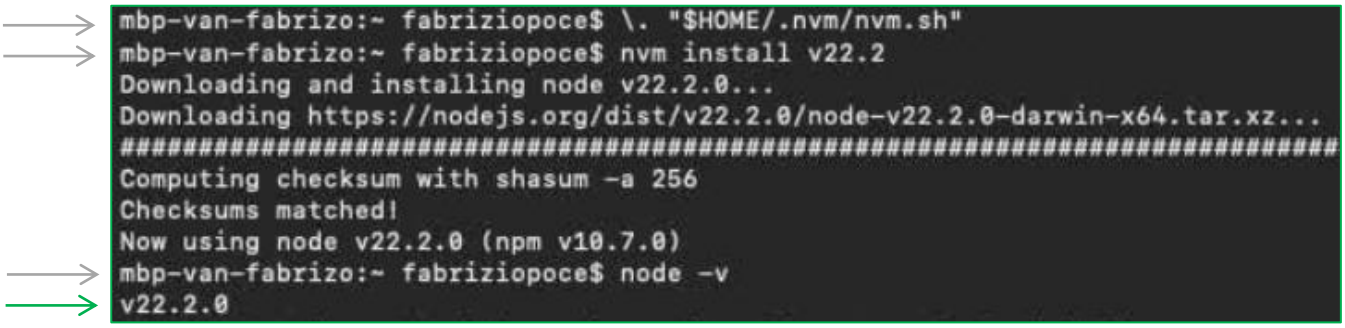
Next start the installation (the compiler will build the binary on your machine):

nvm install 22

Finally, after the installation we verify the Node.js version:

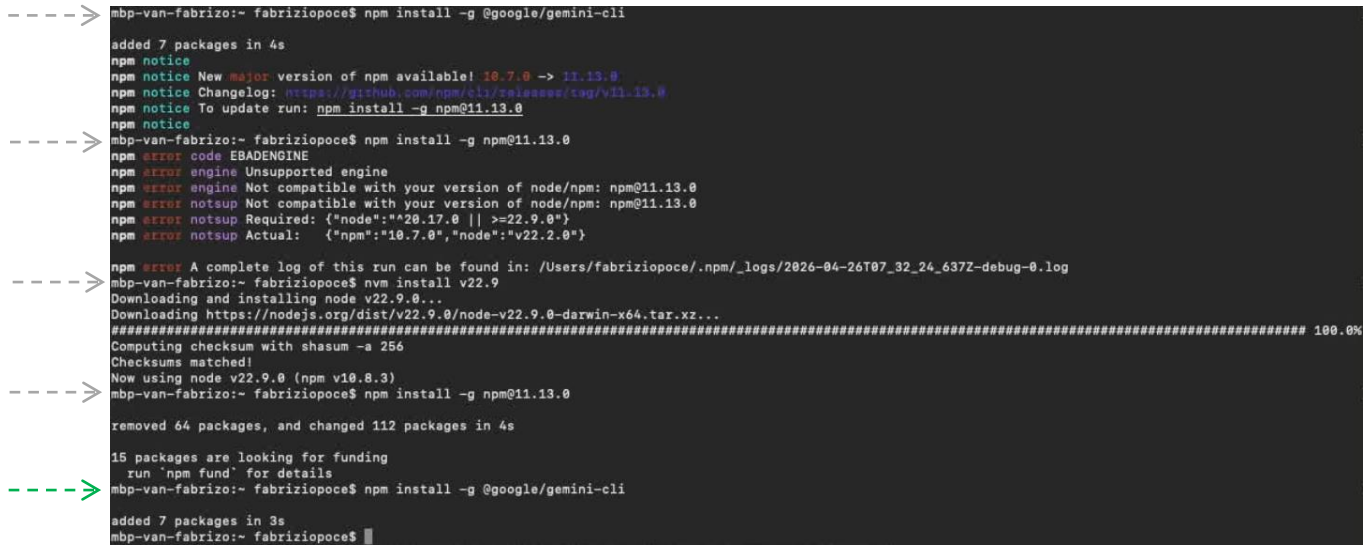
node -v

If it all works, it should look like this (the arrow highlight the commands been given):



Bingo! Node.js runs on your Mac OS system now and you can install Google Gemini CLI, using Google instructions from https://geminicli.com/docs/get-started/installation/).

Extra: When installing Gemini CLI you might be asked other minor updates (to either Node.js npm or nvm modules), like in the picture below (that is: after installing Node.js 22.2.0, Google Gemini CLI installation command suggested Node.js a later npm module, hence the update to Node.js 22.9.0). See picture below (dashed arrows for the commands been given).



It is actually not strictly required, but it is probably best to do it as Google asks for it.