J74 Assistant Developer Tools / OSC Bridge Overview

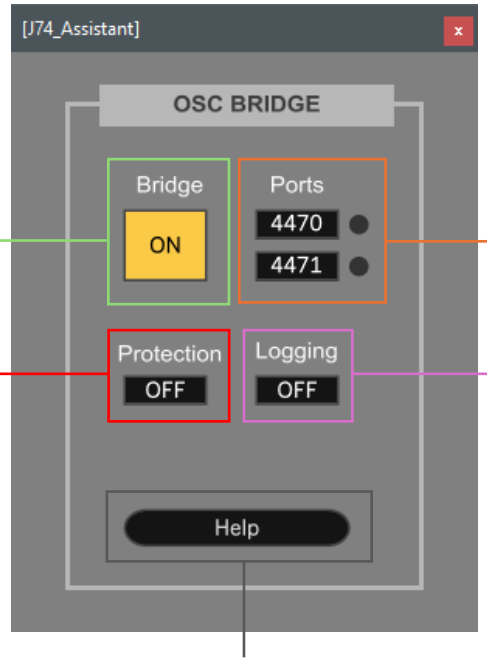
The J74 Assistant Developer Tools package provides tools for external programming of Ableton Live, covering API connection, methods, rules and example prompts for the use of external AI Tools capable of running code on the local machine and for direct scripting and automation of Ableton Live. The following use cases are covered by the package:

[1] the use of the provided companion python scripting environment for automation of Ableton Live actions (snapshots, backups, MIDI operations, warp markers operations)
[2] the use of a “local-coding-enabled” AI Tool (such as Gemini CLI) as a direct-access, multi-purpose “assistant” capable of creating and manipulating an Ableton Live project directly

IMPORTANT: To use the toolset, the OSC Bridge must be present n the Ableton Live project. It is a transparent audio device and can be loaded on any track, including the Master track.

[**Bridge ON/OFF**] Activate (ON) / deactivate (OFF) the OSC Bridge. The bridge allows applications to communicate with the LOM API inside Ableton Live. External applications can be the python scripting toolset (provided with the package), a “local-coding-enabled” AI Tool trained for the communication with the OSC Bridge plugin (e.g. Gemini CLI coupled with the provided toolset and instruction set) or even a third-party tool which follows the OSC Bridge environment protocol rules.

[**Bridge Protection**] If enabled (ON) it prevents write operations from external tools. This option is disabled (OFF) by default.



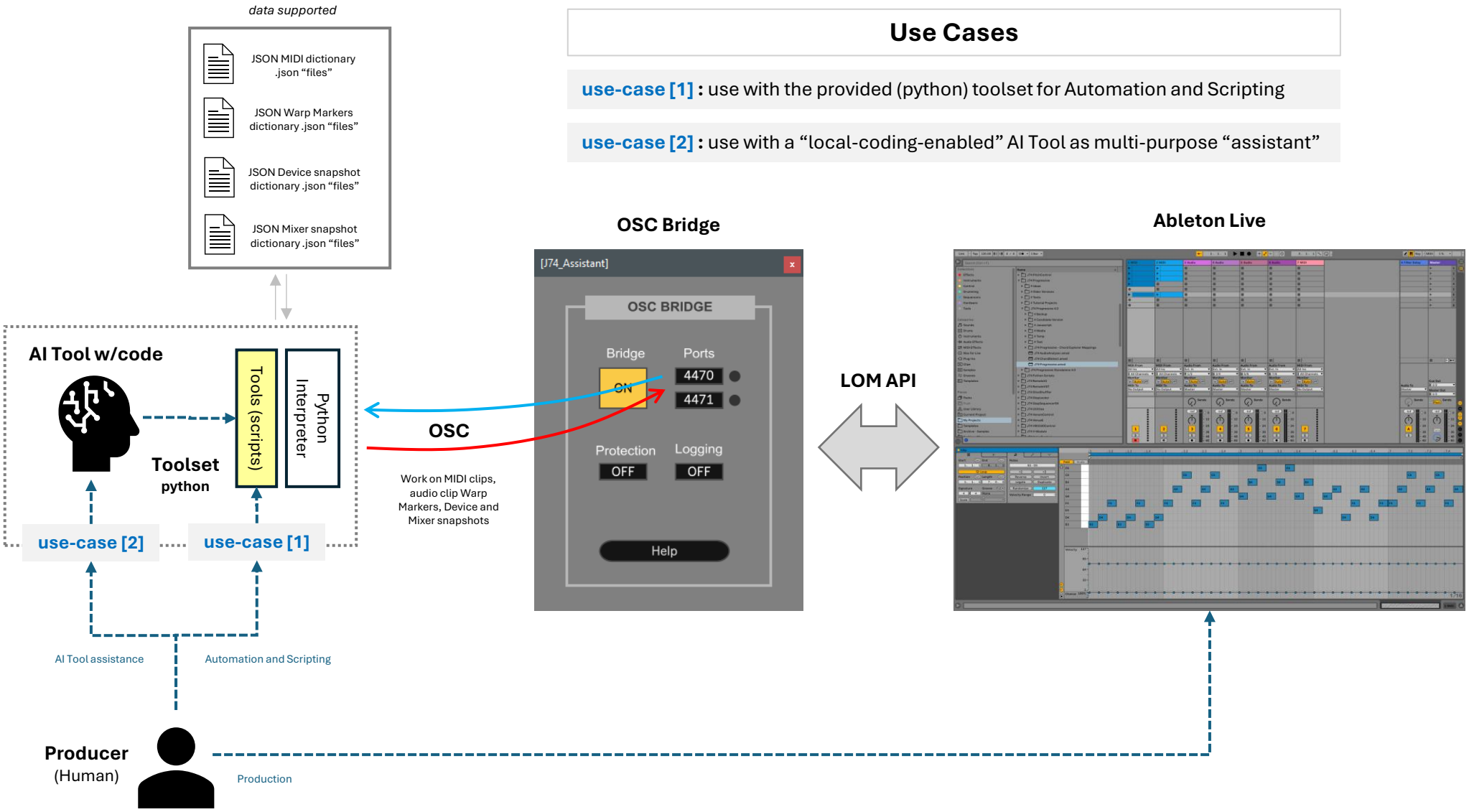
[**Input OSC/UDP port**] The port used on the localhost for incoming CMDs (from the external application to the OSC Bridge). If not necessary, leave on the default port 4470.

[**Output OSC/UDP port**] The port used on the localhost for outgoing CMDs (from the OSC Bridge to the external application). If not necessary, leave on the default port 4471.

[**Logging**] If enabled (ON) logs all messages handled by the Bridge. Useful for troubleshooting. This option is disabled (OFF) by default.

[**Help**] Opens the help section.

J74 Assistant Developer Tools: Use-Cases



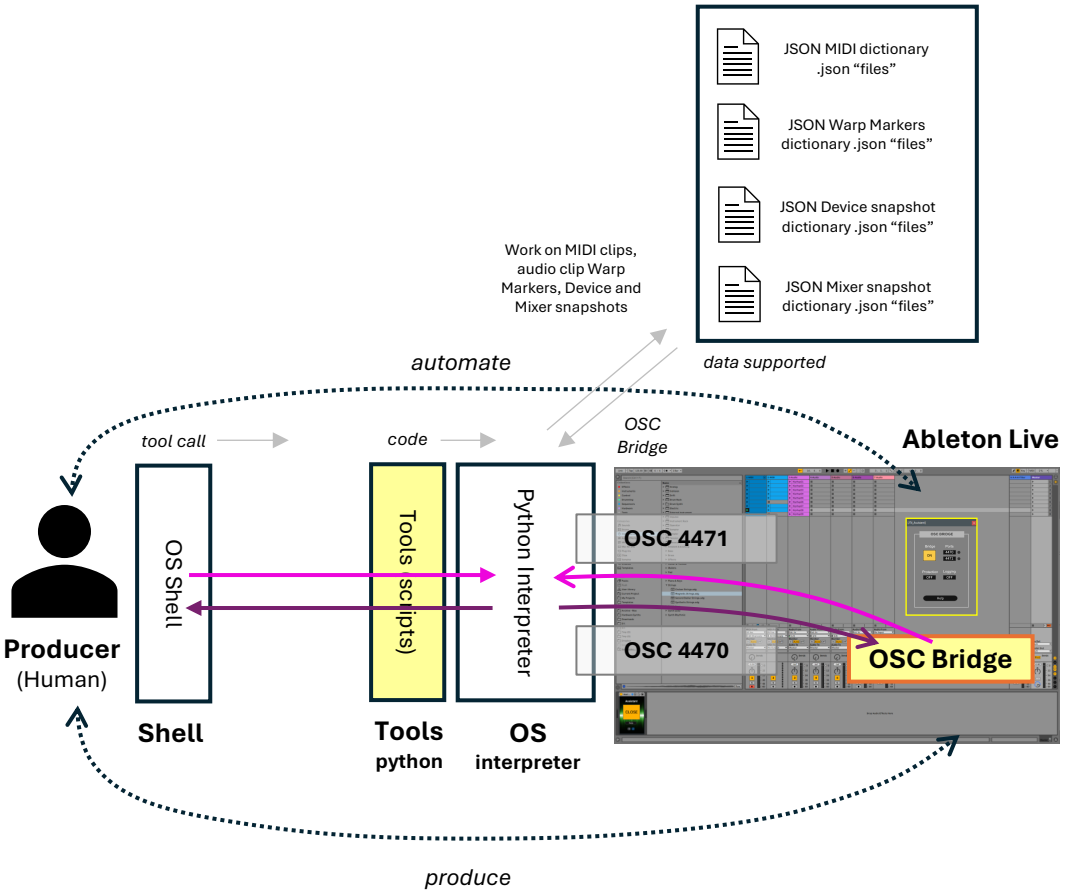
Use Cases

use-case [1] : use with the provided (python) toolset for Automation and Scripting

use-case [2] : use with a "local-coding-enabled" AI Tool as multi-purpose "assistant"

Use-Case [1]: Use with the provided (python) toolset for Automation and Scripting

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



The OSC Bridge allows external applications to communicate with the LOM API inside Ableton Live. In this use case the "application" is the python toolset provided with the package. The OSC Bridge controls and adapts command calls.

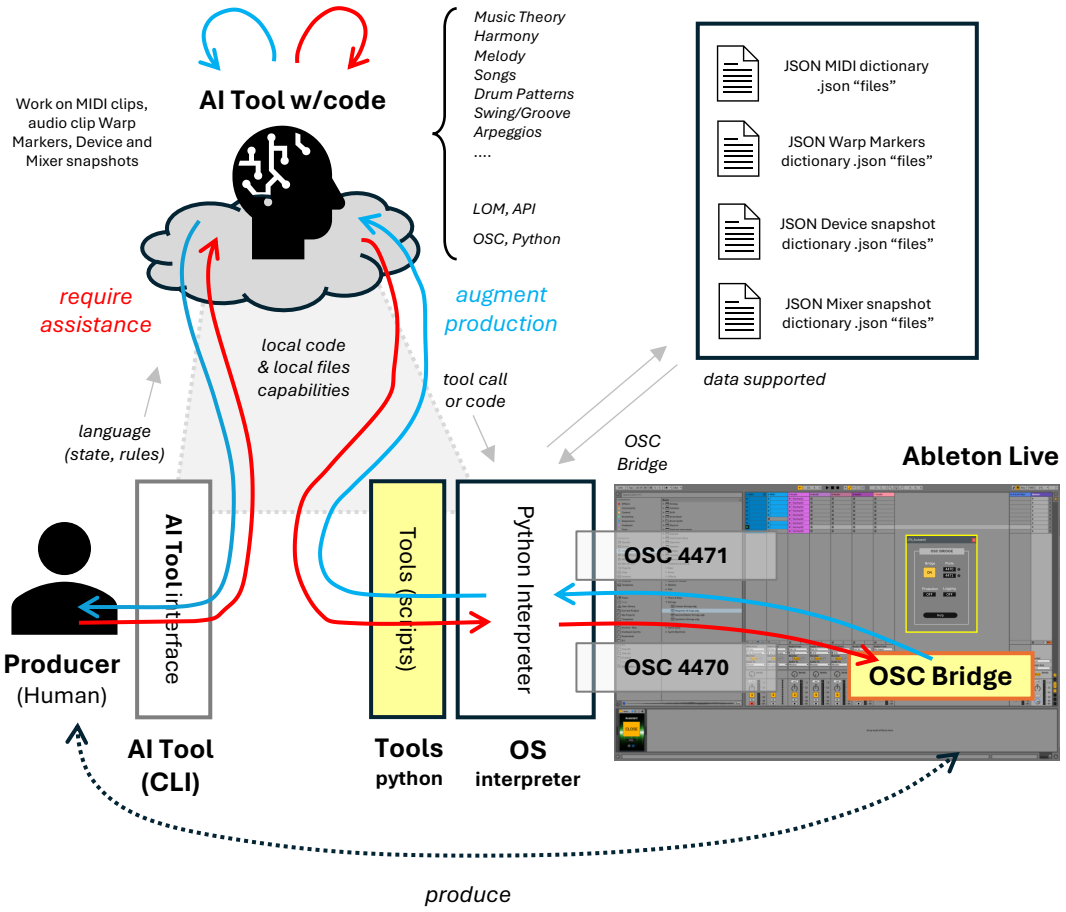
REMARK: see the [Appendix/1] in the documentation for details on how to use the python toolset part of the package.

IMPORTANT: see the "**developer**" section in the documentation for details on how to use the package in this scenario.

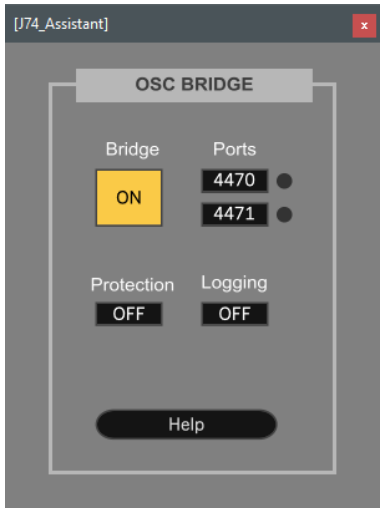
IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

Use-Case [2]: Use with a “local-coding-enabled” AI Tool as multi-purpose “assistant”

[2] “local-coding-enabled” AI tools (Gemini CLI, Claude Code) as multi-purpose assistant



The OSC Bridge allows external applications to communicate with the LOM API inside Ableton Live. In this use case the application is a “local-coding-enabled” AI Tool (such as Gemini CLI), trained for communication with the OSC Bridge plugin and, through the methods in the provided python toolset, for use of the LOM API.



REMARK: see the further in the documentation for details on how to integrate a “local-coding-enabled” AI Tool.

IMPORTANT: see the “**developer**” section in the documentation for details on how to use the package in this scenario.

IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

[DEV notes]: What is covered in this guideline

Background: The J74 Assistant Developer Tools package provides a connection point (OSC Bridge) *and* a python scripting environment. This combination provides the functionality of a MCP (Model Context Protocol) server for an “local-coding-enabled” AI Tool (an AI Tool capable of running and modifying local python code as well as work on local files). This way you can use such AI Tool to control *directly* an Ableton Live project (as use-case[2]). The same connection point (OSC Bridge) *and* a python scripting environment is also suitable to automation and scripting *use-case[1]), for instance for procedural creation of extremely large live sets, backup and restore device or mixer snapshots, perform device or mixer state interpolation and so forth.

Setting up the integration of these software elements requires some system integration.
This section addresses this aspect of the setup.

Content of the following slides:

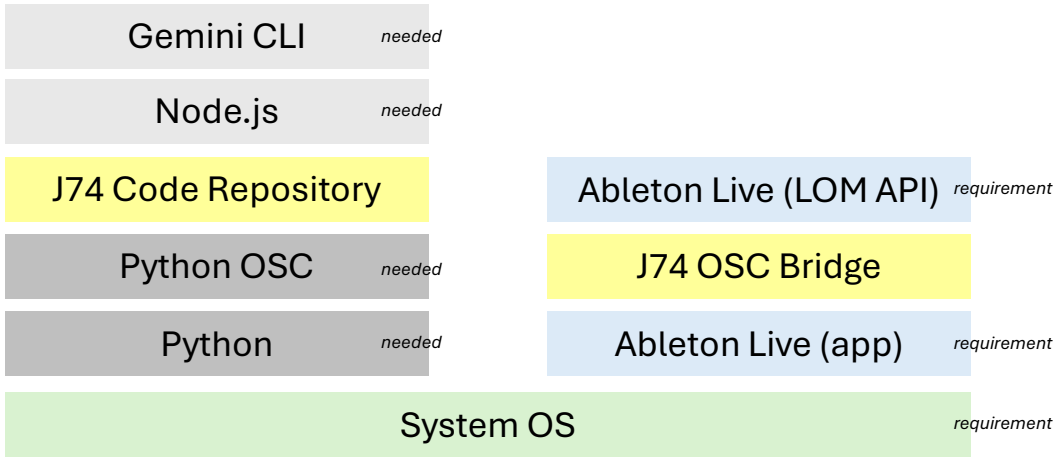
- The Software Stack for this package
- Data Types used by the package
- Protocols and APIs used in the environment
- The Python toolset, which is part of the package
- The MCP Framework for a “local-coding-enabled” AI Tool, also part of the package

- Setup information for Gemini CLI
- Prompts examples for a “local-coding-enabled” AI Tool
- Transparent Actions [pass-through LOM operations] for a “local-coding-enabled” AI Tool
- Guided Actions for MIDI Clips (create, manipulate, etc.) for a “local-coding-enabled” AI Tool
- Guided Actions for Audio Clips (warp markers processing) for a “local-coding-enabled” AI Tool
- Scripted Actions for Device & Mixer snapshots (interpolate, restore, etc.) for a “local-coding-enabled” AI Tool

- Limitations (aka: not possible in the API LOM == not possible for the package)
- What is currently accessible via the LOM API (Ableton Live 11.0 and above)
- The architecture of the J74 OSC Bridge device

[DEV notes]: Software Stack

Here some notes on the software stack built using the “J74 Assistant Developers Tools” package.



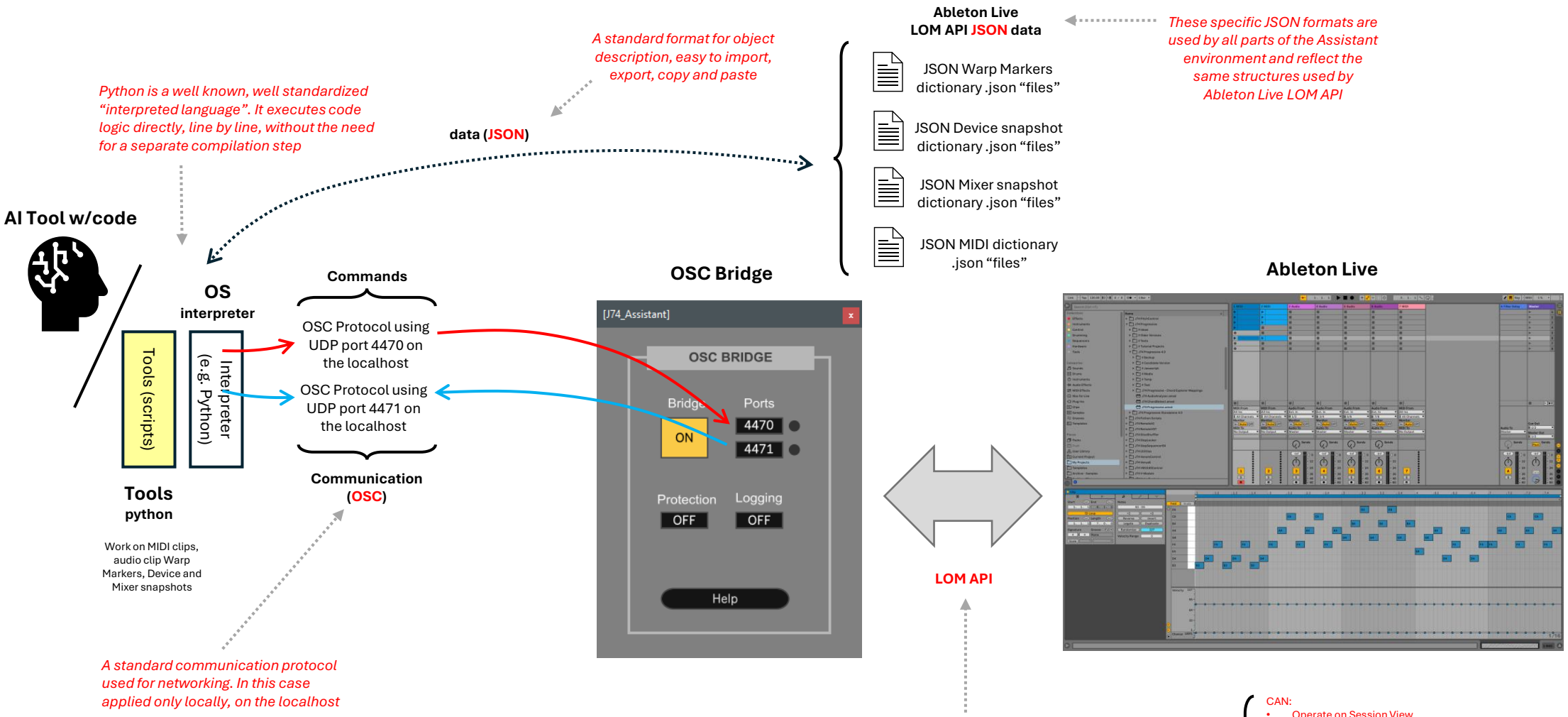
The “J74 Assistant Developer Tools” package is a set bridging technologies, which makes two, otherwise separate worlds, talk the same language: an AI Tool and Ableton Live. It therefore requires some system integration to let all the layers talk to each other (although no additional coding is required).

The installation and Setup process covered by the user manual describes in detail what is needed to get this running.

About the parts in this stack and their requirements:

- Requirement: Compatible OS system (Windows 10 or 11, MAC OS 11 or later - minimum 11 [Big Sur], recommended 14 [Sonoma])
- Requirement: Ableton Live 11 or 12, with Max for Live (as in Ableton Live “Suite” or with add-on license from Ableton)
- This package: an “OSC Bridge” Max for Live plug-in & the “Code Repository” (both parts of this package)
- Needed: Python environment, the Python interpreter and its “python-osc” library (both installed as part of the setup process)
- Needed: AI Tool software tools, Node.js as platform and Gemini CLI as actual AI tool on top (both installed as part of the setup process)

[DEV notes]: Data Types, Protocols, API's



The "J74 Assistant Developer Tools" package is a set bridging technologies, which makes two, otherwise separate worlds, talk the same language: an AI Tool and Ableton Live. Here a summary of the technology involved, the Data Types (JSON), the Protocols (OSC) and the API's.

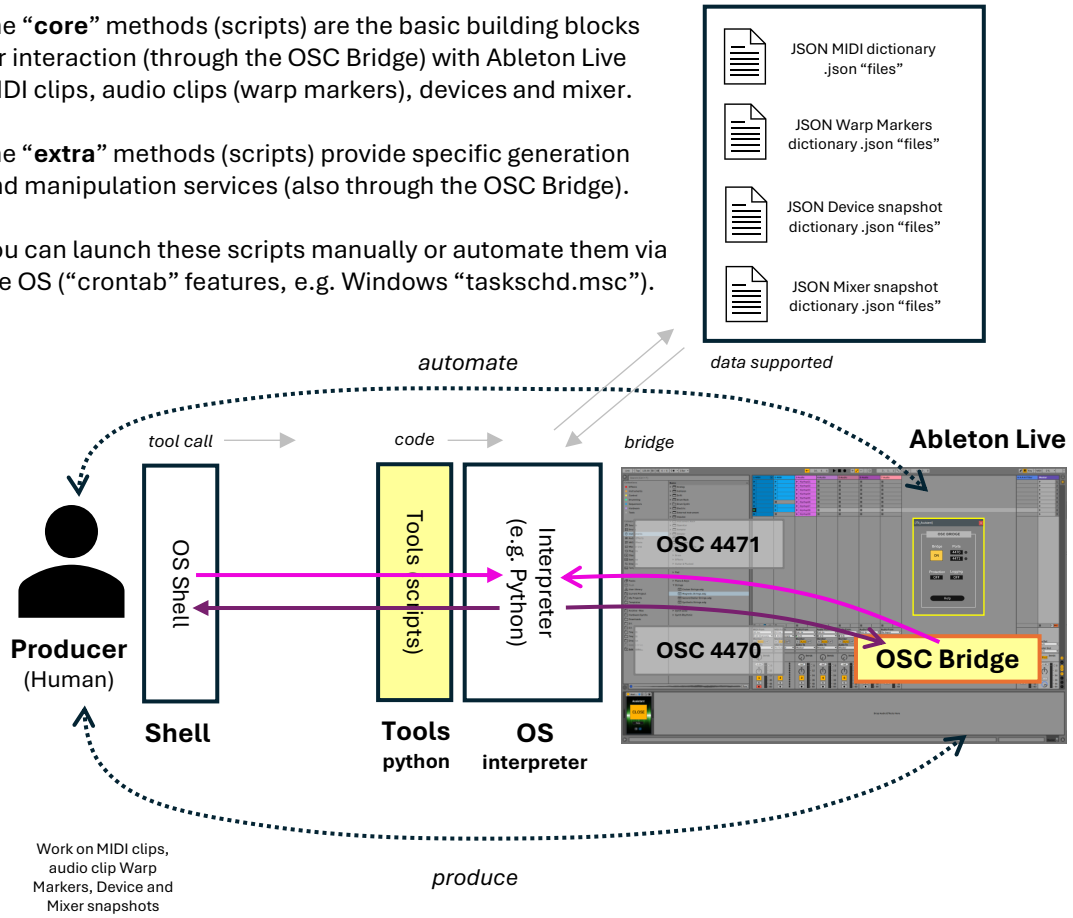
[DEV notes]: Use of the python toolset for automation (requires the bridge to be active/1)

The python toolset provided with the package requires a **python** interpreter (on MAC OS a standard feature, on Windows a downloadable feature) and provides shell (command line) scripts for a variety of operations.

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

The "**extra**" methods (scripts) provide specific generation and manipulation services (also through the OSC Bridge).

You can launch these scripts manually or automate them via the OS ("crontab" features, e.g. Windows "taskschd.msc").



core

[a] python core/collect_midi_clip.py <TRACK> <SLOT>
[b] python core/create_midi_clip.py <TRACK> <SLOT> <JSON_file>
[c] python core/collect_warp_markers.py <TRACK> <SLOT>
[d] python core/create_warp_markers.py <TRACK> <SLOT> <JSON_file>
[e] python core/move_warp_marker_relative_distance.py <TRACK> <SLOT> <MARKER> <VALUE>
[f] python core/device_snapshot.py <TRACK> <SLOT>
[g] python core/restore_device_snapshot.py <TRACK> <SLOT> <JSON_file>
[h] python core/mixer_snapshot.py <OPTION>
[i] python core/restore_mixer_snapshot.py <JSON_file> <OPTION>
[j] python core/clear_warp_markers.py <TRACK> <SLOT>
[w] python core/verify_loopback.py
[z] python core/lom_command_osc.py

extra

- midi_clip_gen
- midi_clip_op
- midi_json_gen
- midi_json_op
- midi_json_tool
- warp_clip_gen
- warp_clip_op
- warp_json_gen
- warp_json_op
- device
- mixer

python generate_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>
python generate_melody_prob.py <ROOT> <SCALE> <PROG> <INTERVAL> <PROBABILITY> <OPTION>
python generate_progression.py <ROOT> <SCALE> <PROG>
python generate_progression_with_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>

python modify_midi_prob.py <JSON> <PROBABILITY> <OPTION>
python modify_midi_timing.py <JSON> <DELAY> <OPTION1> <OPTION2>
python modify_midi_vdev.py <JSON> <VDEV_VALUE>
python arpeggiate_json.py <JSON> <INTERVAL>

python extra/midi_json_tool/midi_analyze.py <JSON_file>

python generate_grid_markers.py <TRACK> <SLOT> <STEP_LENGTH> <SWING_AMOUNT>

python randomize_device_params.py <TRACK> <DEVICE_NUMBER>

python interpolate_mixer_snapshots.py <JSON_file> <JSON_file>

Subset of the extra methods

IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

[DEV notes]: Use of the python toolset for automation (requires the bridge to be active/2)

In general:

- If you run any tool *without* parameters, it will provide you with an inline help.

```
Usage: python modify_midi_timing.py <JSON> <DELAY> <OPTION1> <OPTION2>
# OPTION1
<OPTION1>=0 >>>> modify all notes
<OPTION1>=1 >>>> modify only notes not placed at exact beat position
<OPTION1>=2 >>>> modify only notes not placed at the beginning of a bar
<OPTION1>=0 >>>> modify all notes
# OPTION2
<OPTION2>=0 >>>> modify all notes in the same direction
<OPTION2>=1 >>>> modify all notes in alternative direction
<OPTION2>=2 >>>> modify all notes in random direction
```

- Example:

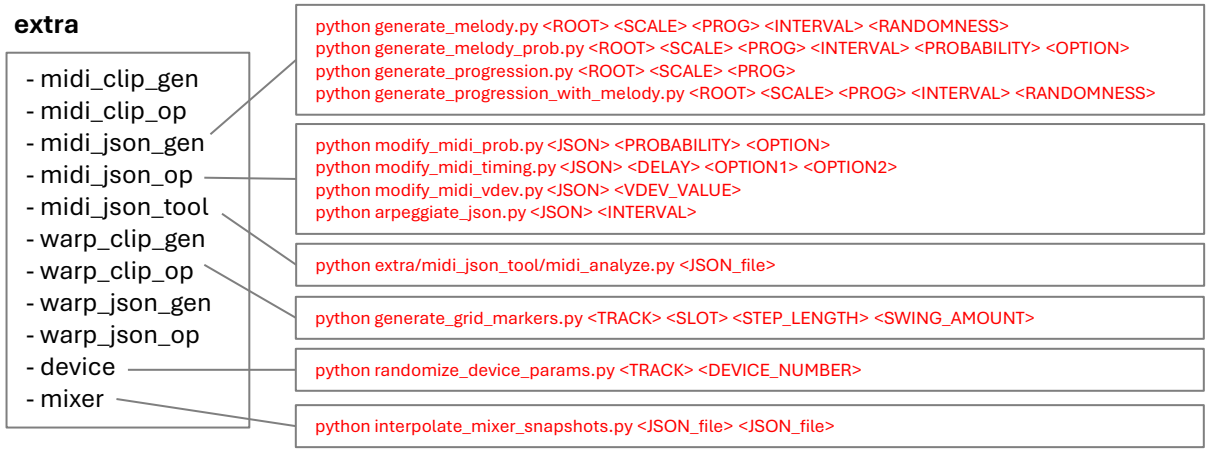
```
python extra/midi_json_op/modify_midi_timing.py
data/MIDI_Clip_Modified_20260413_1048.json 0.25 2 2
```

Please refer to appendix [1] at the end of this document for an explanation of what each python method (script) can do and how to use it.

core

```
[a] python core/collect_midi_clip.py <TRACK> <SLOT>
[b] python core/create_midi_clip.py <TRACK> <SLOT> <JSON_file>
[c] python core/collect_warp_markers.py <TRACK> <SLOT>
[d] python core/create_warp_markers.py <TRACK> <SLOT> <JSON_file>
[e] python core/move_warp_marker_relative_distance.py <TRACK> <SLOT>  <MARKER> <VALUE>
[f] python core/device_snapshot.py <TRACK> <SLOT>
[g] python core/restore_device_snapshot.py <TRACK> <SLOT> <JSON_file>
[h] python core/mixer_snapshot.py <OPTION>
[i] python core/restore_mixer_snapshot.py <JSON_file> <OPTION>
[j] python core/clear_warp_markers.py <TRACK> <SLOT>
[w] python core/verify_loopback.py
[z] python core/lom_command_osc.py
```

extra



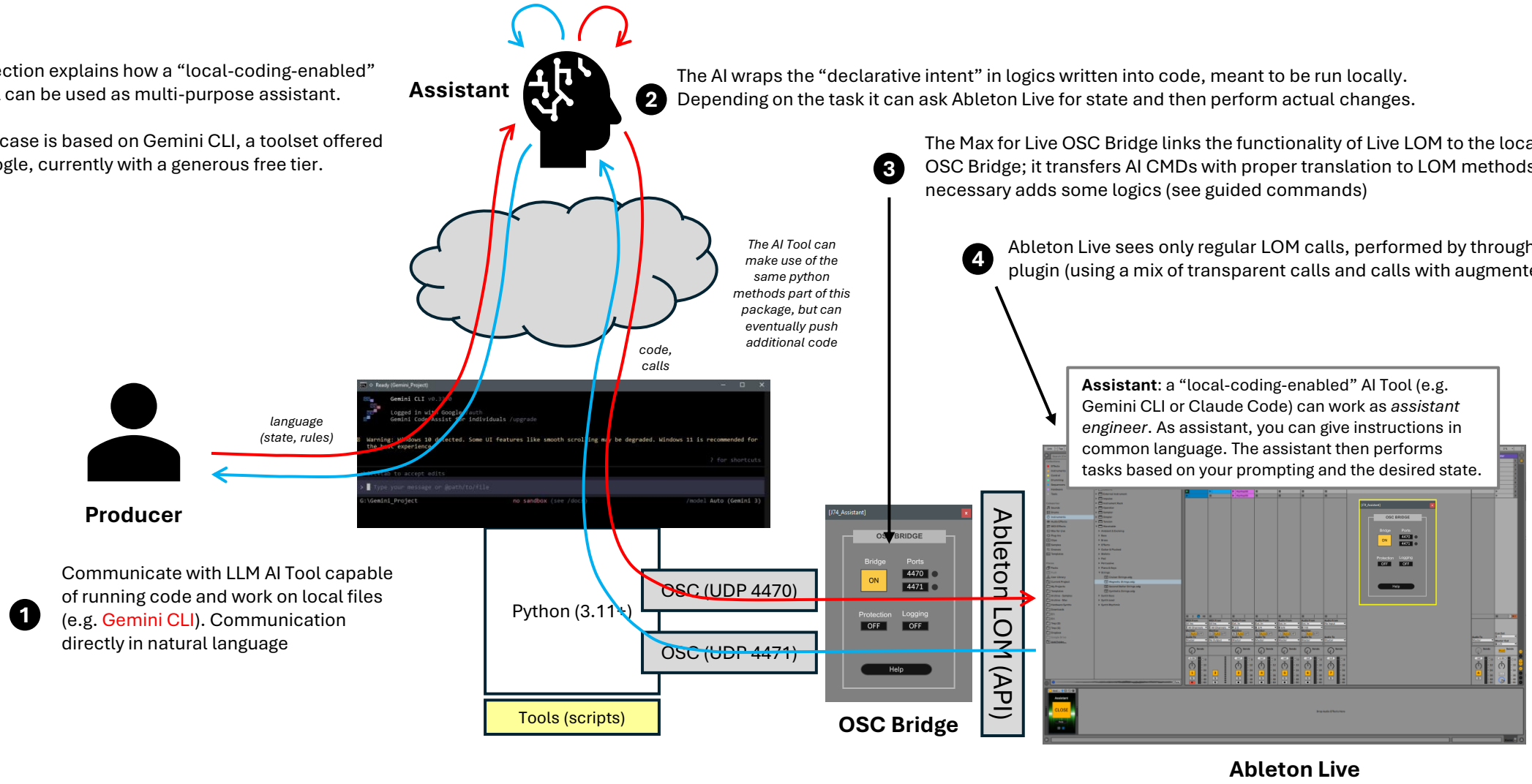
Subset of the extra methods

IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

[DEV notes]: Use of the toolset a MCP Framework for a “local-coding-enabled” AI Tool

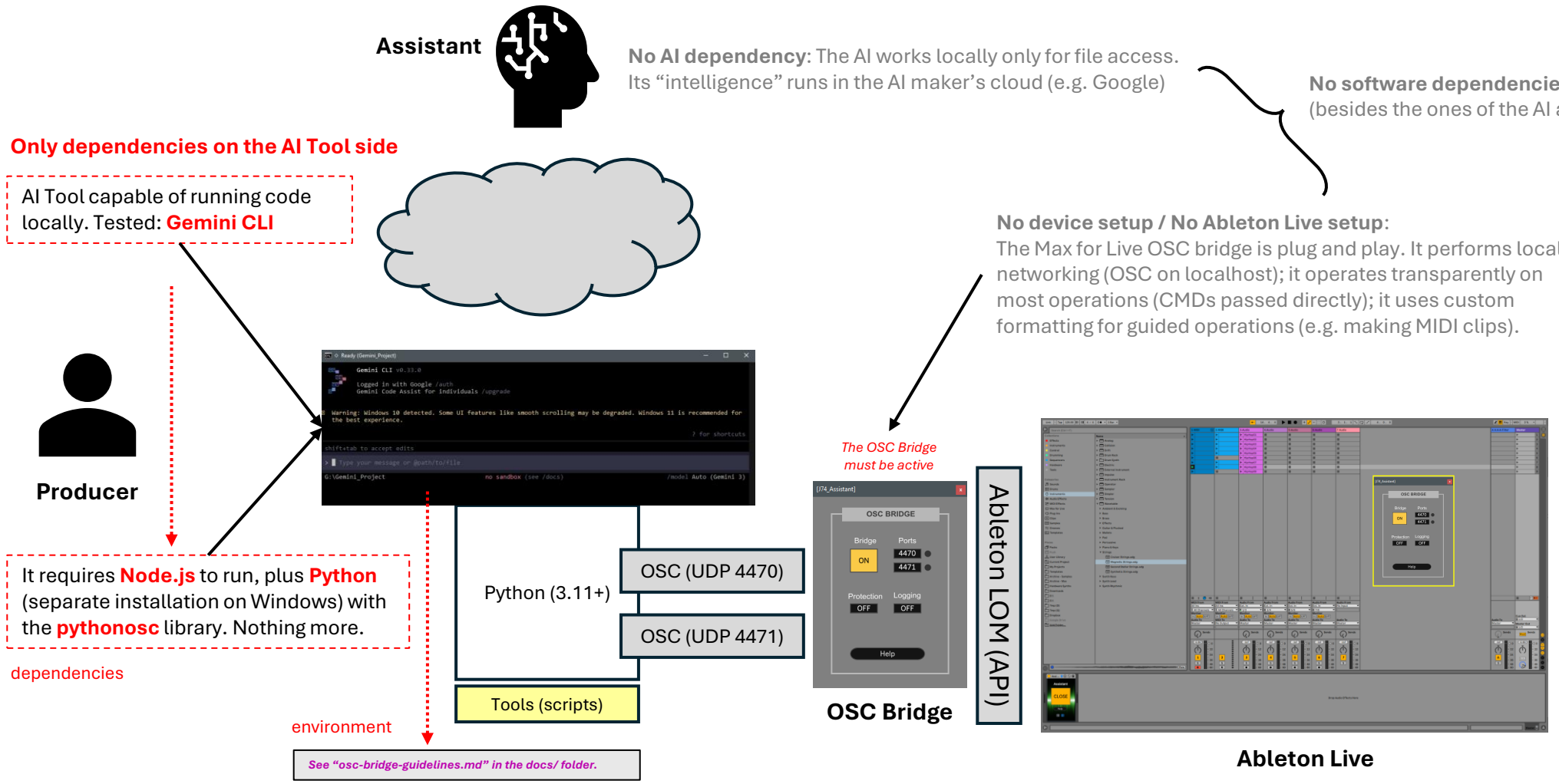
This section explains how a “local-coding-enabled” AI Tool can be used as multi-purpose assistant.

Thuse case is based on Gemini CLI, a toolset offered by Google, currently with a generous free tier.



IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

[DEV notes]: What you need to know for using Gemini CLI with the J74 Assistant Developer Tools package



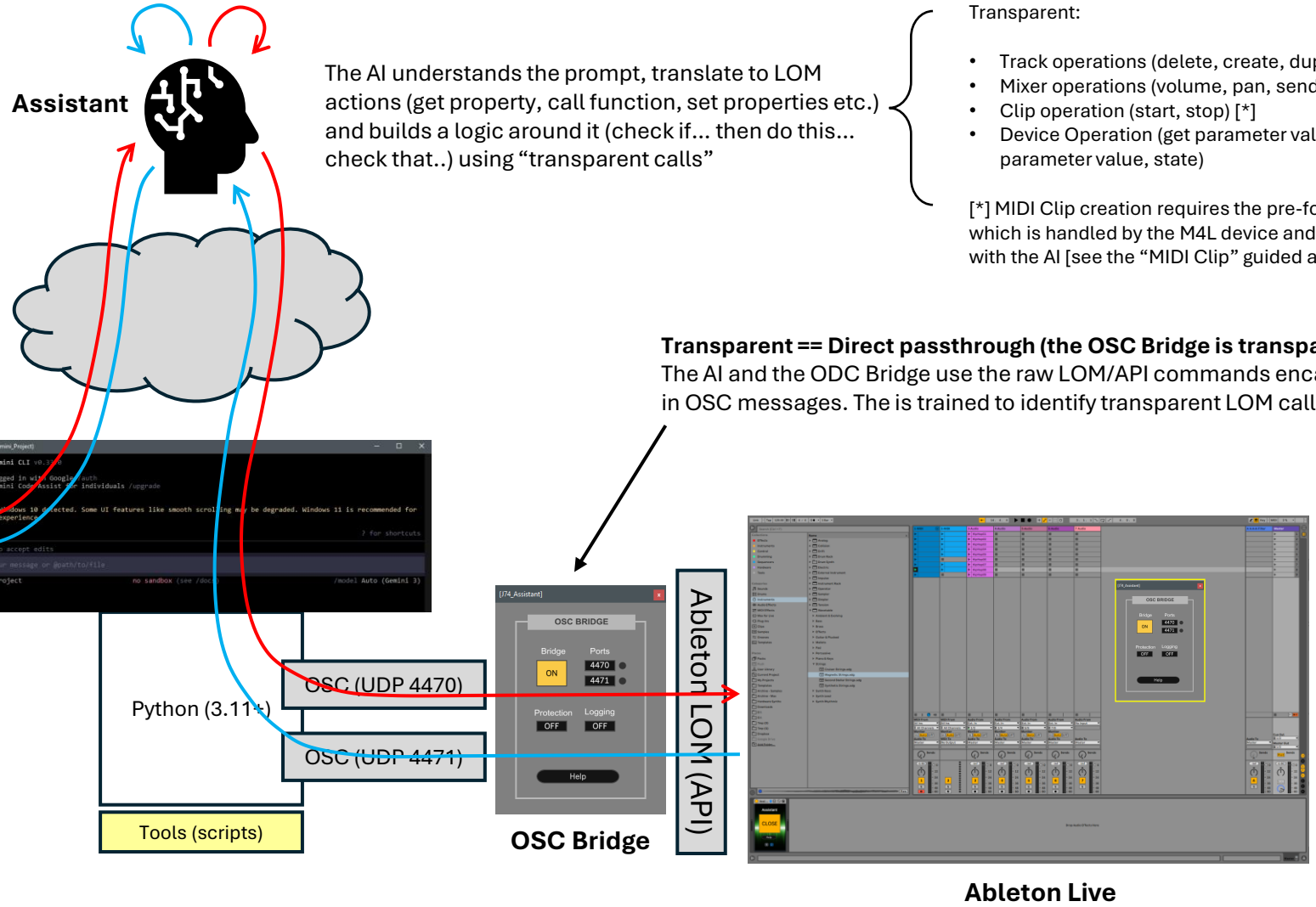
IMPORTANT: The toolset works using OSC communication.
Therefore, the OSC Bridge needs to be enabled.

[DEV notes]: Prompts for a “local-coding-enabled” AI Tool (e.g. Gemini CLI) and their underlying nature

Prompt Examples:

Prompt Examples:	Type of underlying action
> Action: add 4 audio tracks after track 3 > Action: apply send A to 0.25 on the first 4 tracks and 0.5 on tracks 5 to 16 > Action: undo the last change	**Transparent LOM actions on “live_set”** (direct LOM API call, wrapped in some logics given by the prompt)
> Action: modify filter 3 on the first EQ8 device in track 2 with q = 0.8 and frequency 3Khz > Action: disable the second EQ device in track 6	**Transparent LOM actions on “live_set tracks devices parameters”** (direct LOM API call, wrapped in some logics given by the prompt)
> Action: create a midi clip 0 0 with a D minor 7th chord long 4 bars > Action: create a midi clip 0 0 with a progression I V IV II in A major 1 bar per chord > Action: create a midi clip 0 0 with a jazz progression in G minor, being played with some swing > Action: create a midi clip 0 0 with the progression of a typical blues song, expanding chords up to 4 notes	**Guided MIDI clip creations for harmony and melody** (wrapped in some logics given by the prompt + bridge processing)
> Action: create a midi clip on track 2 slot 0 for a drum beat. This drum beat should have a Caribbean flavor sixteen instruments (in typical drum rack layout) make it 4 bars long and variate each bar > Action: create a clip in 1 0 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 same timing of clip in 0 0.	**Guided MIDI clip creations for drumming** (wrapped in some logics given by the prompt + bridge processing)
> Action: modify clip 0 0 and transpose only C# and G# notes a semitone down > Action: harmonize the clip 0 0 to the C minor key > Action: duplicate clip 0 0 (double length, repeat the clip notes) > Action: transform clip 0 0 into a progression where each chord's notes are arpeggiated sequentially at 1/4 bar intervals.	**Guided MIDI clip modify operations (harmonize, arpeggio, etc.)** (wrapped in some logics given by the prompt + bridge processing)
> Action: add warp markers every 1/16 on audio clip 7 of track 3 > Action: remove the 5th and 6th warp markers from clip 7 of track 3 > Action: counting from 0 as first, move all the odd warp markers in clip 7 of track 3 of 1/32 (of a bar)	**Guided Audio clip warp marker operations** (wrapped in some logics given by the prompt + bridge processing)
> Action: collect a device snapshot for the first device on track 3 > Action: collect a device snapshot for the second device on track 3 > Action: interpolate between snapshots {A} and {B} and push the resulting values to the third device on track 3. > Action: randomize the parameters of the fourth device in track 3	**Scripted Device Snapshot operations** (wrapped in some logics given by the prompt + script & bridge processing)
> Action: collect a mixer snapshot for all tracks (excluding returns and master) > Action: collect a mixer snapshot for all tracks > Action: interpolate between mixer snapshots {A} and {B} and push the resulting values to the tracks. > Action: randomize the mixer parameters of the first fourth tracks	**Scripted Mixer Snapshot operations** (wrapped in some logics given by the prompt + script & bridge processing)

[DEV notes]: Transparent Actions [pass-through LOM operations]



The AI understands the prompt, translate to LOM actions (get property, call function, set properties etc.) and builds a logic around it (check if… then do this… check that..) using “transparent calls”

Transparent:

- Track operations (delete, create, duplicate)
- Mixer operations (volume, pan, sends, solo, mute)
- Clip operation (start, stop) [*]
- Device Operation (get parameter value, state, set parameter value, state)

[*] MIDI Clip creation requires the pre-formatted logics which is handled by the M4L device and is pre-shared with the AI [see the “MIDI Clip” guided actions case]

Example Prompt: Delete track 1; Create 32 Audio Tracks and set volume to 0; Create an additional audio track and set volume to -6dB, send A to 0.5.

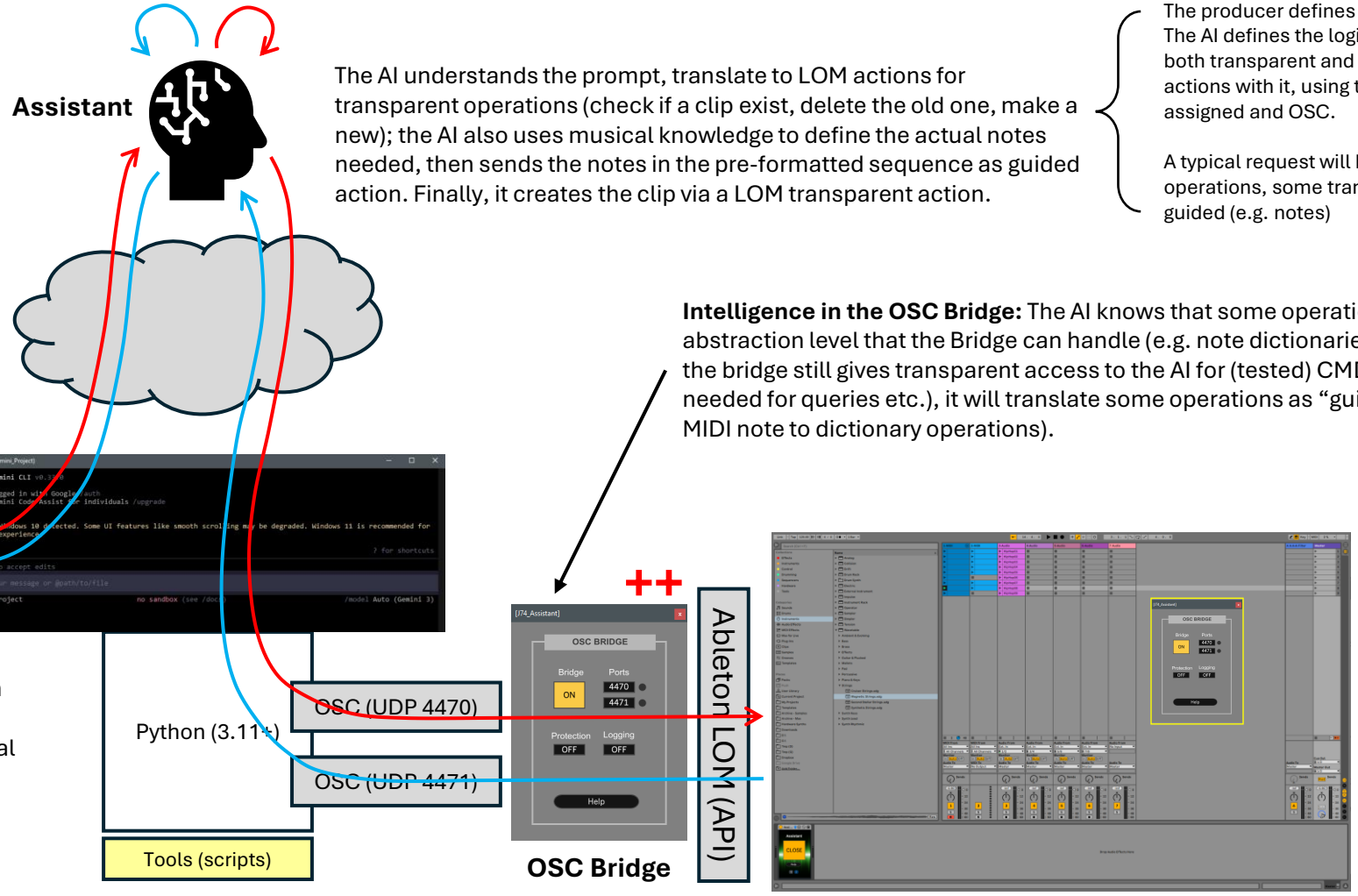
AI knows the rules of communication, the Ableton Live LOM and sends **transparently RAW** LOM CMDs wrapped in logic conform prompting.

Transparent == Direct passthrough (the OSC Bridge is transparent):
The AI and the ODC Bridge use the raw LOM/API commands encapsulated in OSC messages. The is trained to identify transparent LOM calls

The Producer defines “what” needs to be achieved, the AI Tool defines “how” to do things and performs this through a combination of RAW LOM control & guided rules, using OSC as communication channel. This is a “Declarative” approach to agentic task assistance, although safe-guarded.

WARNING: These actions can potentially be disruptive. The user (producer) always remains responsible for the AI tool actions, which potentially can be disruptive if left unchecked.

[DEV notes]: Guided Actions for MIDI Clips (create, manipulate, etc.)



The AI understands the prompt, translate to LOM actions for transparent operations (check if a clip exist, delete the old one, make a new); the AI also uses musical knowledge to define the actual notes needed, then sends the notes in the pre-formatted sequence as guided action. Finally, it creates the clip via a LOM transparent action.

The producer defines the state. The AI defines the logic and wraps both transparent and guided LOM actions with it, using the methods assigned and OSC.

A typical request will have several operations, some transparent, some guided (e.g. notes)

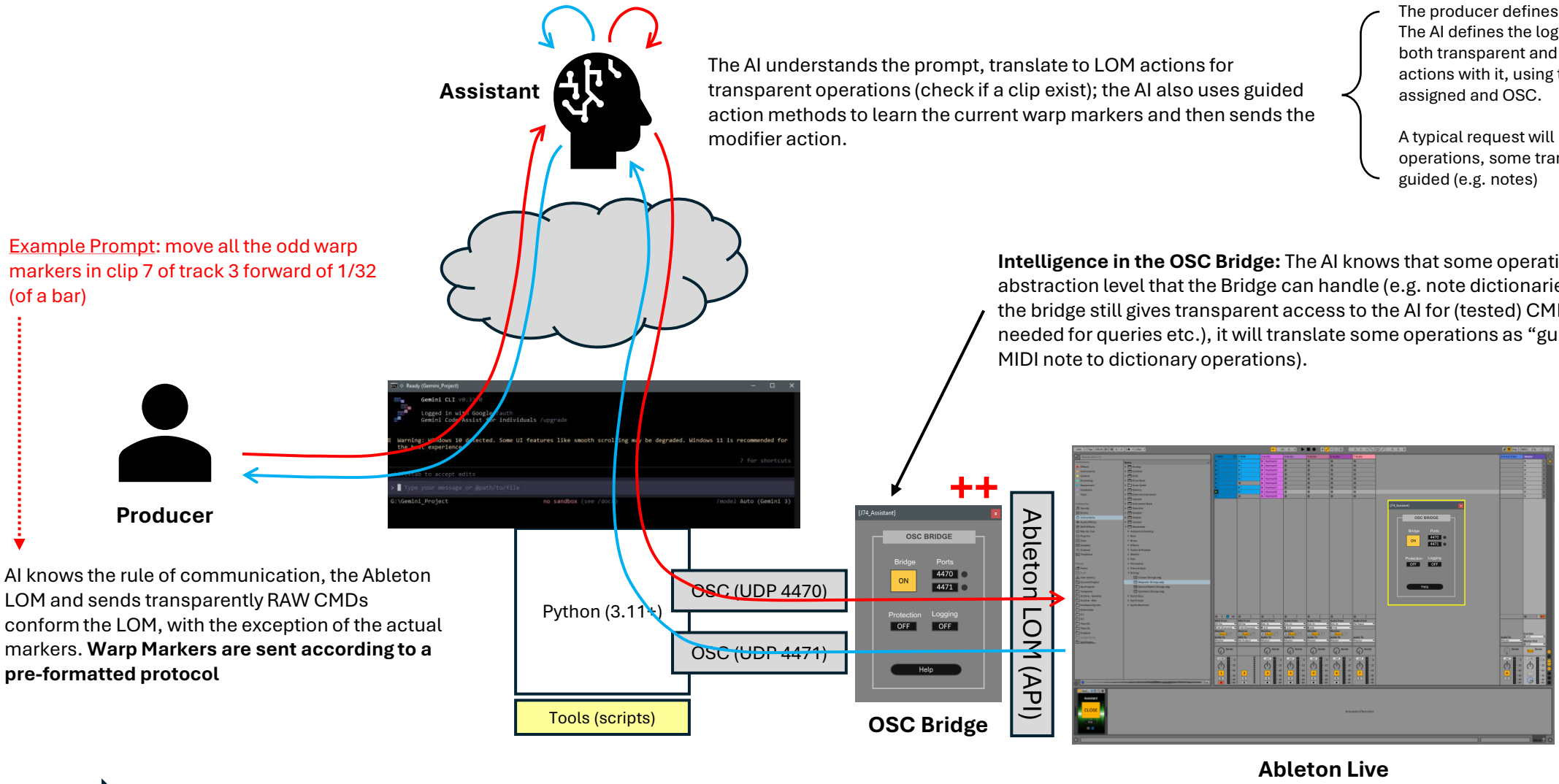
Example Prompt: Create a blues progression in the C minor scale of 12 bars in track 1 slot 1, using sparsely chord extensions. Build a melodic line on top.

Intelligence in the OSC Bridge: The AI knows that some operation need an abstraction level that the Bridge can handle (e.g. note dictionaries). While the bridge still gives transparent access to the AI for (tested) CMDs (e.g. needed for queries etc.), it will translate some operations as “guided” (e.g. MIDI note to dictionary operations).

AI knows the rule of communication, the Ableton LOM and sends transparently RAW CMDs conform the LOM, with the exception of the actual notes. **Notes are sent according to the pre-formatted protocol**

The Producer defines “what” needs to be achieved, the AI Tool defines “how” to do things and performs this through a combination of RAW LOM control & guided rules, using OSC as communication channel. This is a “Declarative” approach to agentic task assistance, although safe-guarded.

[DEV notes]: Guided Actions for Audio Clips (warp markers processing)



The AI understands the prompt, translate to LOM actions for transparent operations (check if a clip exist); the AI also uses guided action methods to learn the current warp markers and then sends the modifier action.

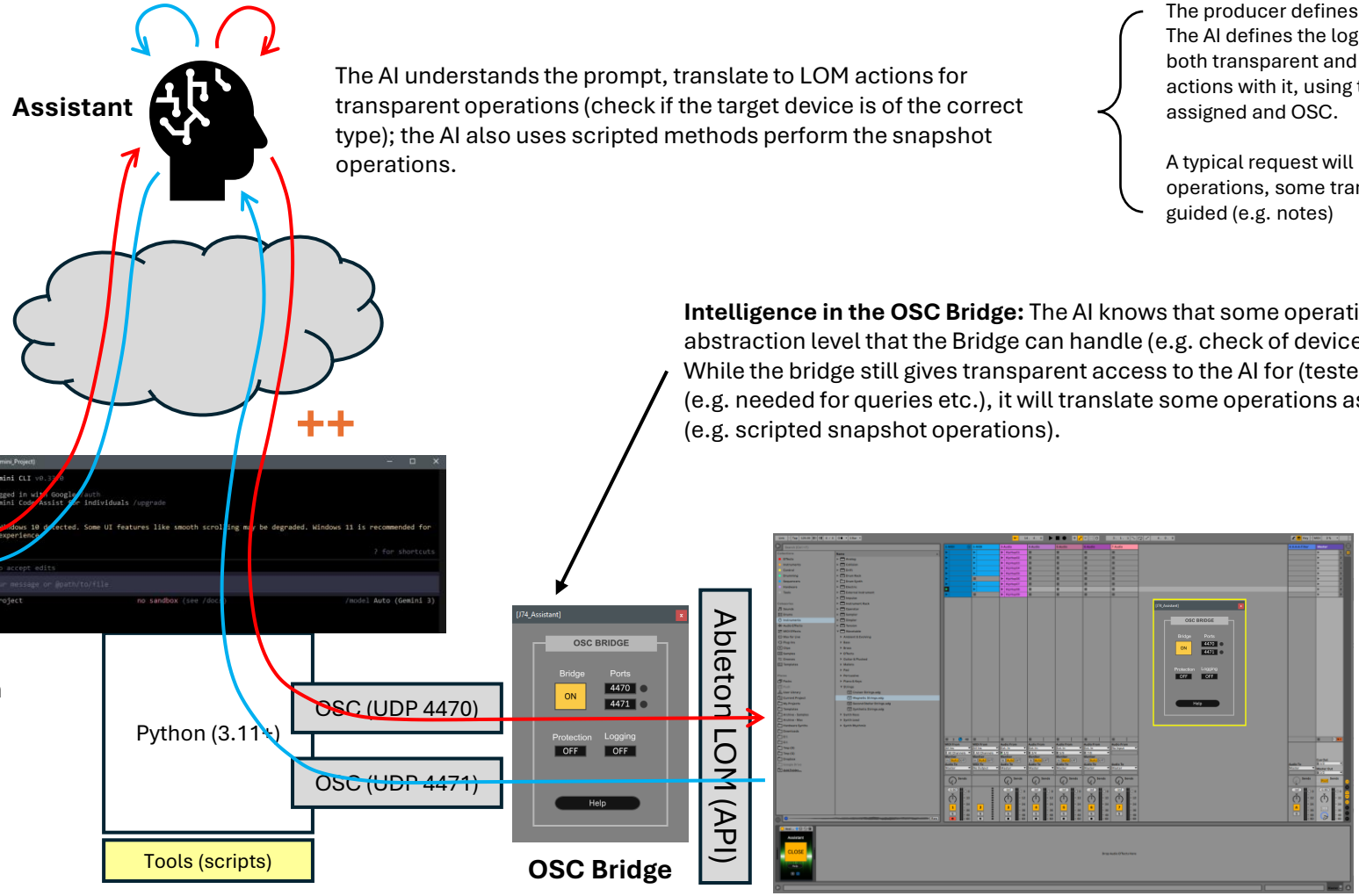
The producer defines the state. The AI defines the logic and wraps both transparent and guided LOM actions with it, using the methods assigned and OSC.

A typical request will have several operations, some transparent, some guided (e.g. notes)

Intelligence in the OSC Bridge: The AI knows that some operation need an abstraction level that the Bridge can handle (e.g. note dictionaries). While the bridge still gives transparent access to the AI for (tested) CMDs (e.g. needed for queries etc.), it will translate some operations as "guided" (e.g. MIDI note to dictionary operations).

The Producer defines "what" needs to be achieved, the AI Tool defines "how" to do things and performs this through a combination of RAW LOM control & guided rules, using OSC as communication channel. This is a "Declarative" approach to agentic task assistance, although safe-guarded.

[DEV notes]: Scripted Actions for Device & Mixer snapshots (interpolate, restore, etc.)



The AI understands the prompt, translate to LOM actions for transparent operations (check if the target device is of the correct type); the AI also uses scripted methods perform the snapshot operations.

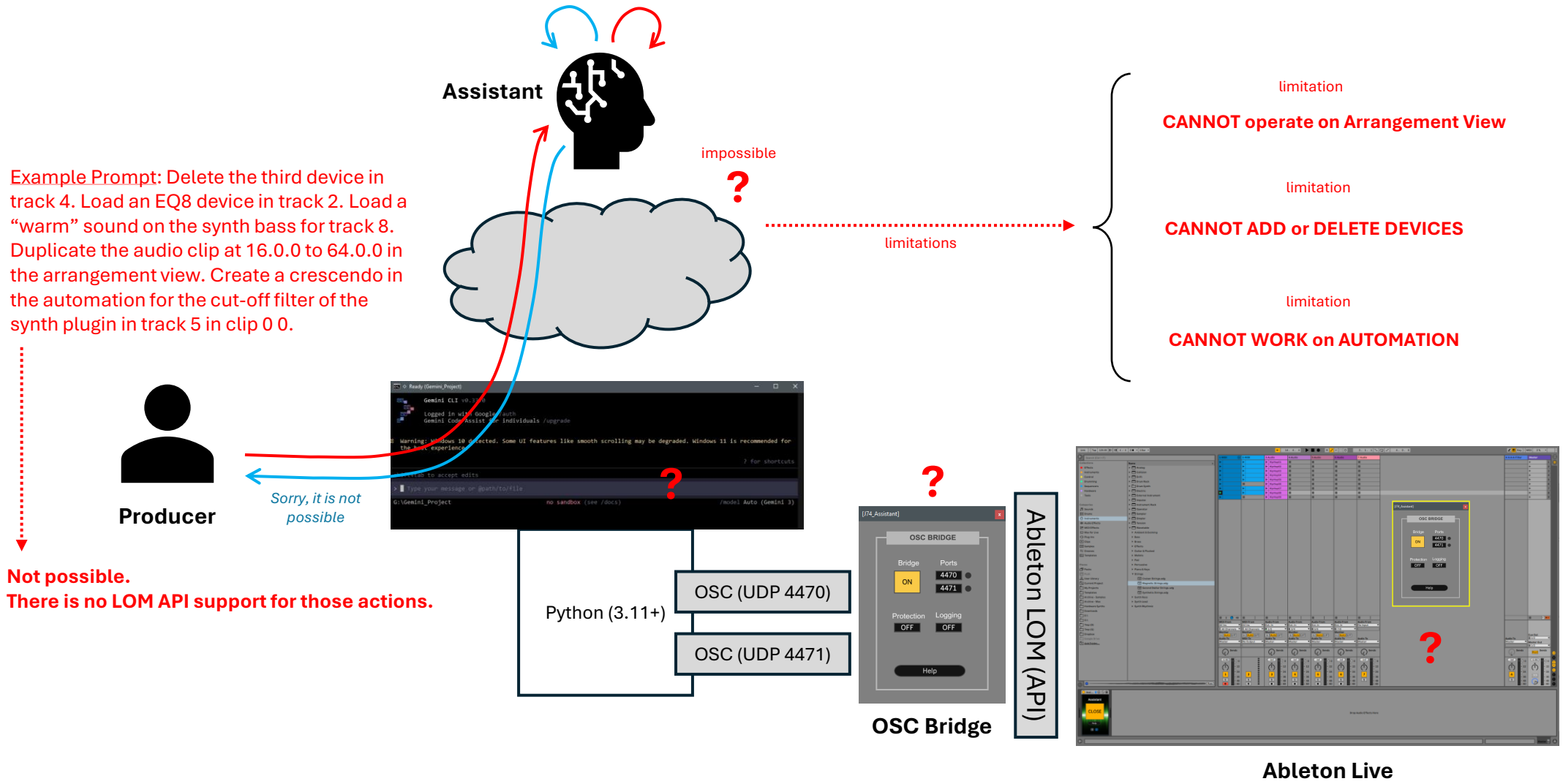
The producer defines the state.
The AI defines the logic and wraps both transparent and guided LOM actions with it, using the methods assigned and OSC.

A typical request will have several operations, some transparent, some guided (e.g. notes)

Intelligence in the OSC Bridge: The AI knows that some operation need an abstraction level that the Bridge can handle (e.g. check of device type). While the bridge still gives transparent access to the AI for (tested) CMDs (e.g. needed for queries etc.), it will translate some operations as “guided” (e.g. scripted snapshot operations).

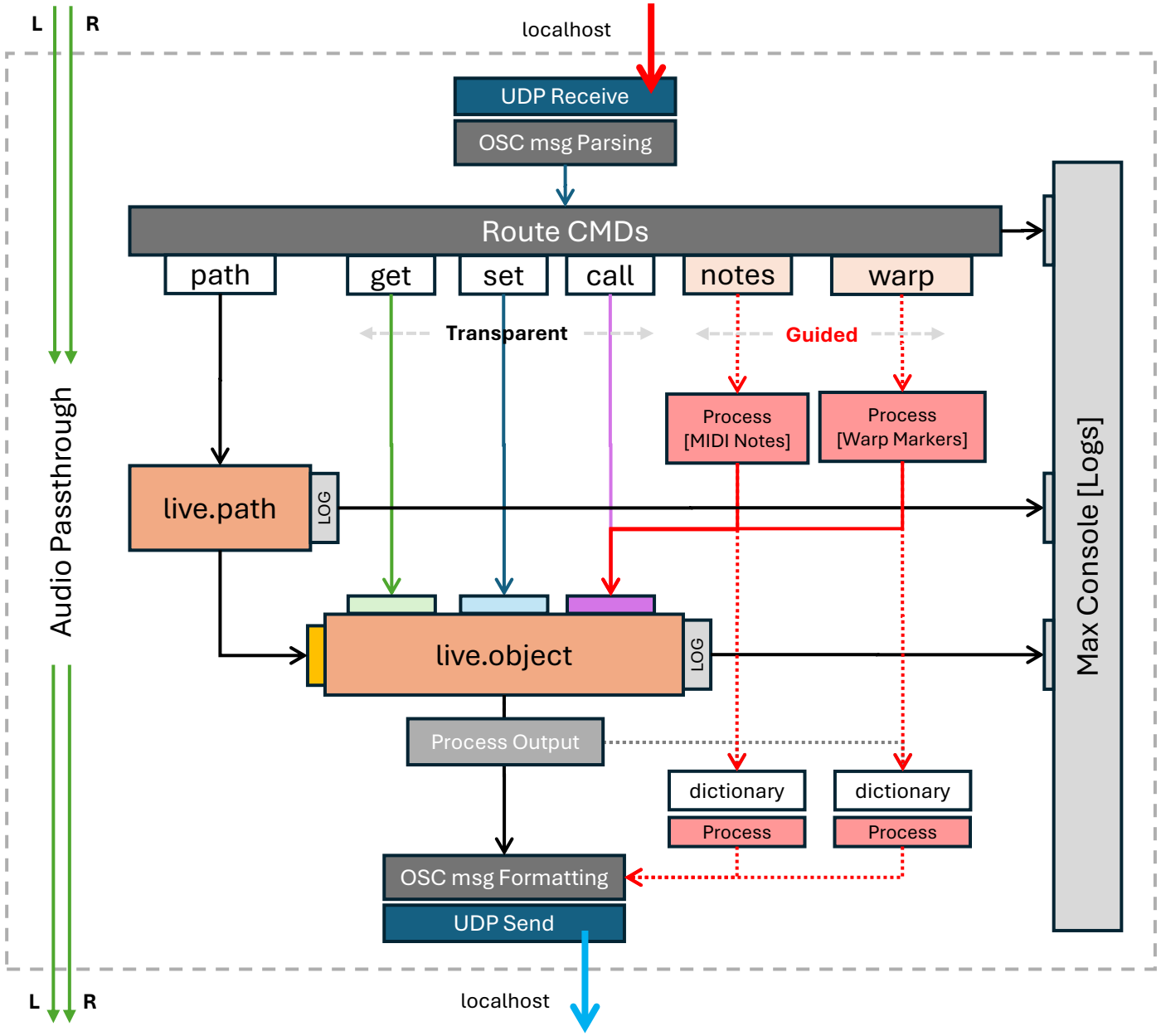
The Producer defines “what” needs to be achieved, the AI Tool defines “how” to do things and performs this through a combination of RAW LOM control & guided rules, using OSC as communication channel. This is a “Declarative” approach to agentic task assistance, although safe-guarded.

[DEV notes]: Limitations (aka: not possible in the API LOM == not possible for the package)



not possible in Ableton Live API LOM == not possible for the package & the AI

[DEV notes]: Architecture of the J74 OSC Bridge device



Concept/1 – the plug (the OSC Bridge)
A custom device you drop on your Master track, that "catches" OSC/UDP CMDs and uses them for the LOM actions to manipulate Ableton Live. When needed (see concept/2), it adds additional logic (e.g. MIDI notes, warp markers, snapshots, dictionaries). It can act as an MCP gateway. It provides COPY & PASTE facilities.

Concept/2 – A plug-n-play environment (Code Repository) for AI Tools
A streamlined environment for an AI tool with pre-defined context, methods, rules, example codes and example prompts. It makes an AI tool immediately ready to go.

NO other dependencies
- The device talks the Ableton/LOM/API natively and interfaces with OSC natively.
- There is no external or cloud MCP or JSON-RPC dependency.
- Other than an AI Tool itself (as the “assistant”) no third-party software is involved.

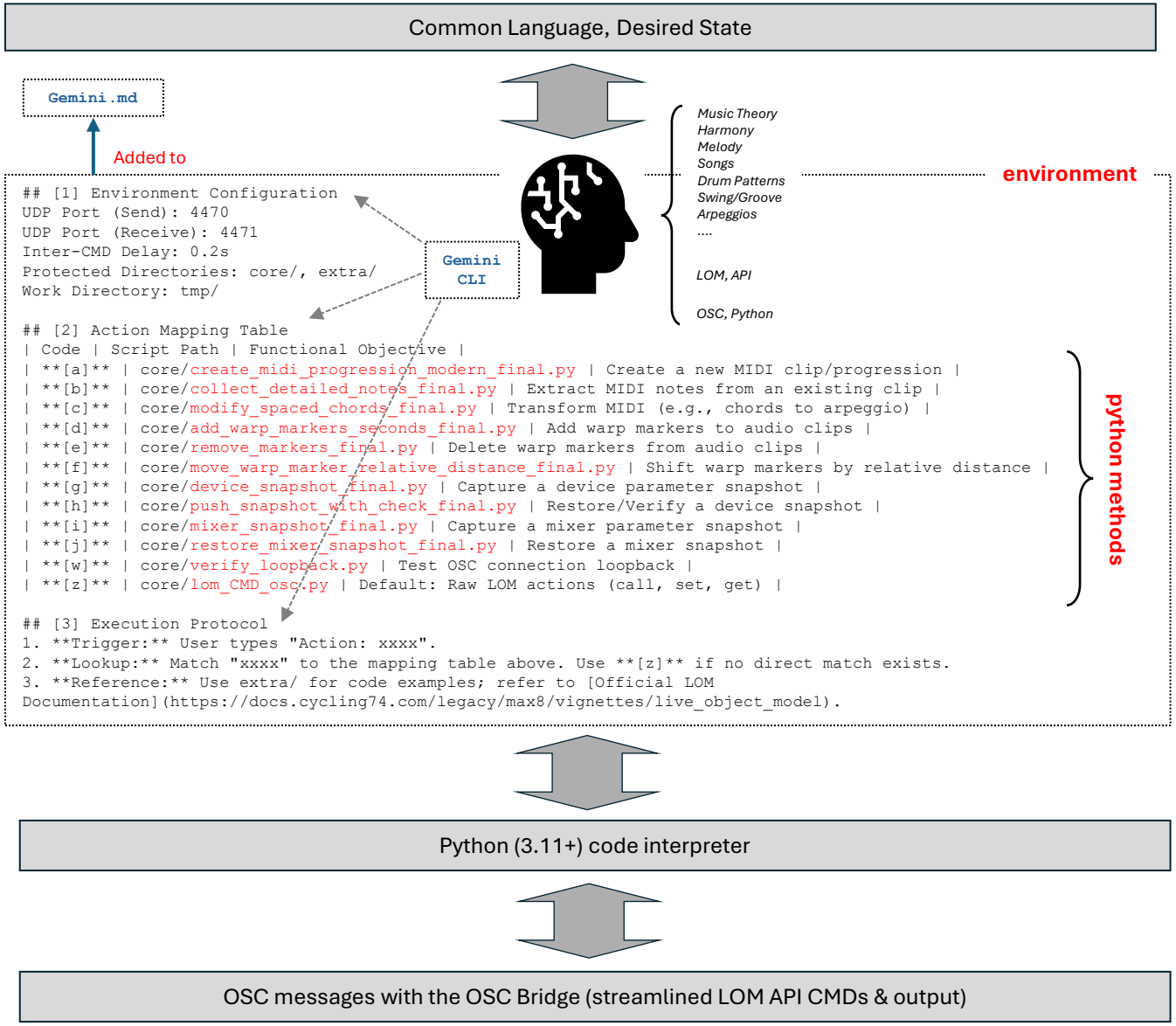
Why an Assistant? [Why do we use an LLM AI model these days?]:
- Because the AI knows “many things” (musical things: harmony, rhythm, whatever)
- Because the AI knows Ableton Live too and its inner mechanisms (LOM API)
- Because this way the producer can use this knowledge defining a desired state
- Because the way the AI can assist and that make the desired state happen

Prompting made easy: goals (a “desired state”) expressed in musical language:
It is a “declarative” (desired state) approach, and the environment provides the AI all the rules, methods and examples it needs to do things. The producer only uses the common language to describe a goal (action). The AI then knows which method to use and wraps actions in logics producing code automatically. The producer does not have to deal with coding at all.

Flexible: any AI Tool can potentially work as an assistant
- Tested and developed with Google Gemini CLI (recommended).
- In theory any LLM can be used *proven* it can run code locally on the host
- The Bridge device can also be used for automation and scripting (no AI)

Safety Disclaimer: *an AI has potentially full Ableton Live LOM API powers on the live set. So, it is the Producer that is responsible for the safety of the live set project, and it is the Producer that needs to guide the AI...*

[DEV notes]: Environment for use with “local-coding-enabled” AI Tools (e.g. Gemini CLI)



Concept/1 – the plug (the OSC Bridge)
A custom device you drop on your Master track, that "catches" OSC/UDP CMDs and uses them for the LOM actions to manipulate Ableton Live. When needed (see concept/2), it adds additional logic (e.g. MIDI notes, warp markers, snapshots, dictionaries). It can act as an MCP gateway. It provides COPY & PASTE facilities.

Concept/2 – A plug-n-play environment (Code Repository) for AI Tools
A streamlined environment for an AI tool with pre-defined context, methods, rules, example codes and example prompts. It makes an AI tool immediately ready to go.

NO other dependencies
- The device talks the Ableton/LOM/API natively and interfaces with OSC natively.
- There is no external or cloud MCP or JSON-RPC dependency.
- Other than an AI Tool itself (as the “assistant”) no third-party software is involved.

Why an Assistant? [Why do we use an LLM AI model these days?]:
- Because the AI knows “many things” (musical things: harmony, rhythm, whatever)
- Because the AI knows Ableton Live too and its inner mechanisms (LOM API)
- Because this way the producer can use this knowledge defining a desired state
- Because the way the AI can assist and that make the desired state happen

Prompting made easy: goals (a “desired state”) expressed in musical language:
It is a “declarative” (desired state) approach, and the environment provides the AI all the rules, methods and examples it needs to do things. The producer only uses the common language to describe a goal (action). The AI then knows which method to use and wraps actions in logics producing code automatically. The producer does not have to deal with coding at all.

Flexible: any AI Tool can potentially work as an assistant
- Tested and developed with Google Gemini CLI (recommended).
- In theory any LLM can be used *proven* it can run code locally on the host
- The Bridge device can also be used for automation and scripting (no AI)

Safety Disclaimer: *an AI has potentially full Ableton Live LOM API powers on the live set. So, it is the Producer that is responsible for the safety of the live set project, and it is the Producer that needs to guide the AI...*

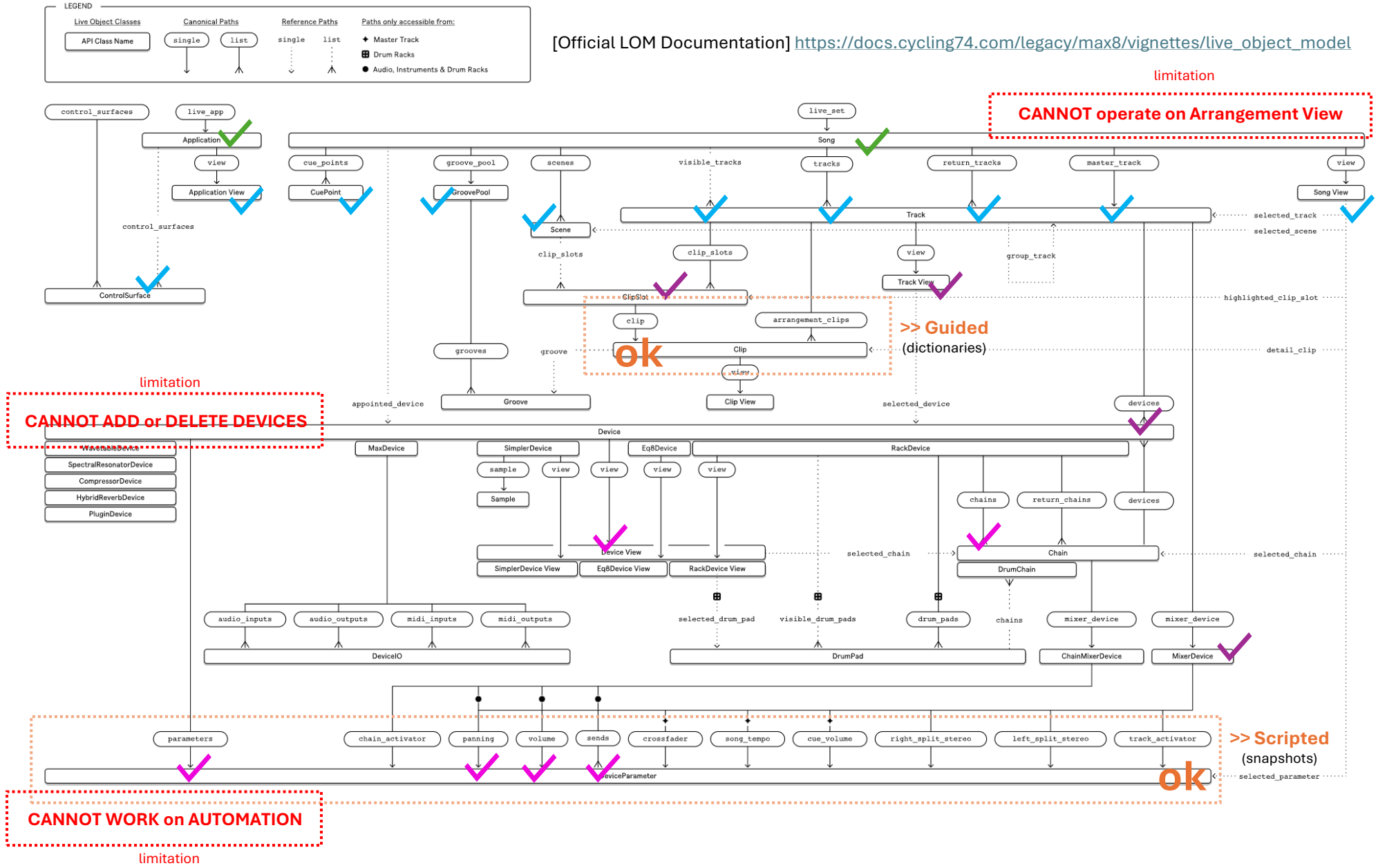
[DEV notes]: What is currently accessible via the LOM API (Ableton Live 11.0 and above)

Transparent

- ✓ Level-0
- ✓ Level-1
- ✓ Level-2
- ✓ Level-3

Guided / Scripted

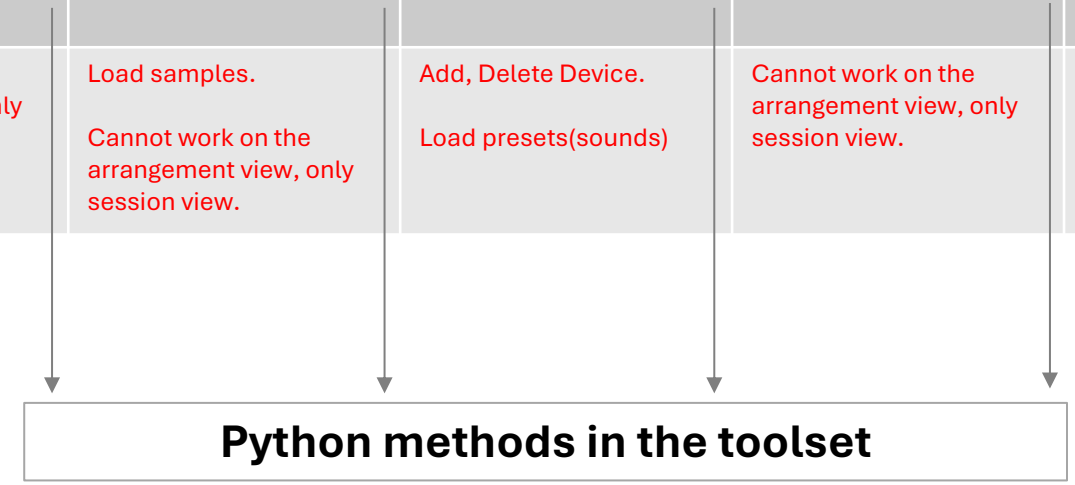
- X Level-0
- X Level-1
- X Level-2
- ok Level-3



[DEV notes]: What is currently accessible via the LOM API (Ableton Live 11.0 and above)

[Official LOM Documentation] https://docs.cycling74.com/legacy/max8/vignettes/live_object_model

	MIDI	Warp Markers	Device	Mixer	Automation	Arrangement View
Possible	Read, Create, Delete, Store, Manipulate MIDI notes in MIDI clips.	Read, Create, Delete, Store, Manipulate Warp Markers notes in audio clips.	Read, Store, Manipulate Parameter values in devices. Snapshot operation.	Read, Store, Manipulate Mixer settings. Snapshot operation. Create Tracks, Delete Tracks.	Cannot work on the arrangement view, only session view.	Cannot work on the arrangement view, only session view.
Not possible	Cannot work on the arrangement view, only session view.	Load samples. Cannot work on the arrangement view, only session view.	Add, Delete Device. Load presets(sounds)	Cannot work on the arrangement view, only session view.	Cannot work on any automation (session/clip or arrangement)	Cannot work on the arrangement view, only session view.



not possible in Ableton Live API LOM == not possible for the package & the AI

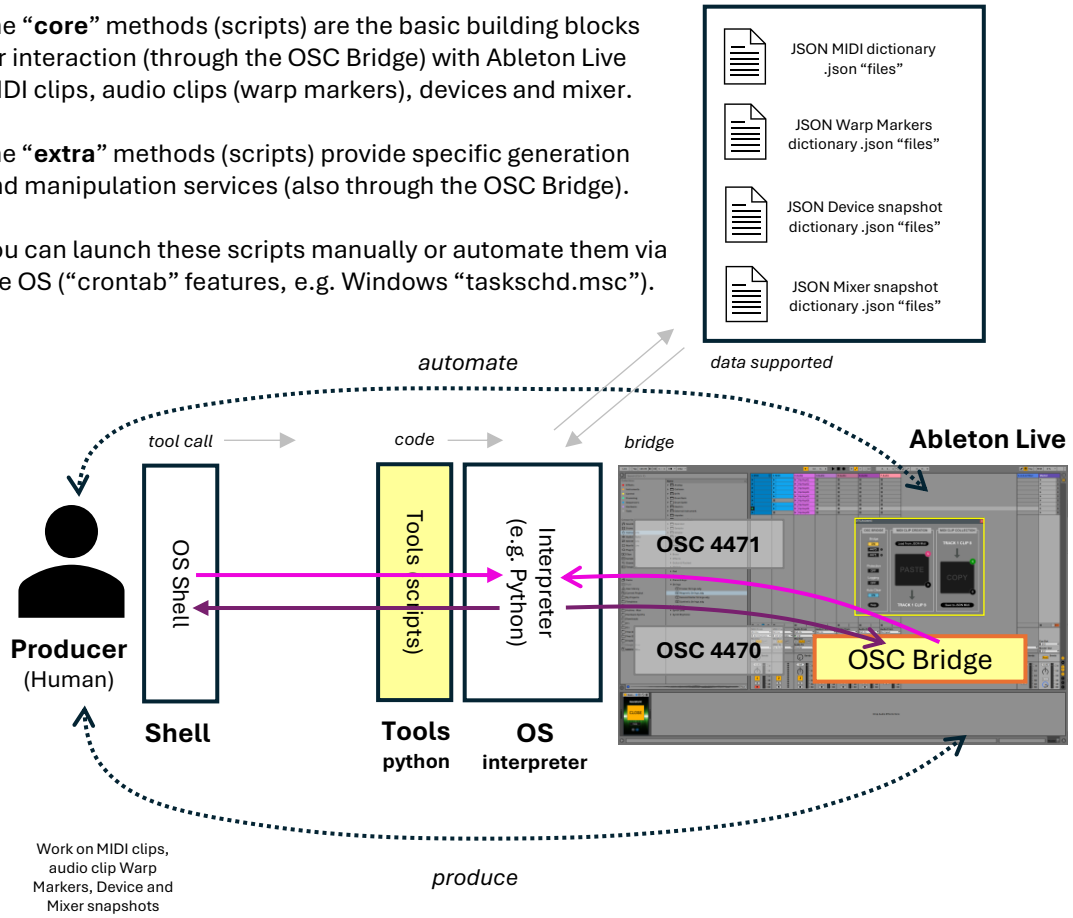
[Appendix/1]: Python methods

This is a toolset provided with the package. It requires a **python** interpreter (on MAC OS a standard feature, on Windows a downloadable feature) and provides shell (command line) scripts for a variety of operations.

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

The "**extra**" methods (scripts) provide specific generation and manipulation services (also through the OSC Bridge).

You can launch these scripts manually or automate them via the OS ("crontab" features, e.g. Windows "taskschd.msc").



core

[a] python core/collect_midi_clip.py <TRACK> <SLOT>
[b] python core/create_midi_clip.py <TRACK> <SLOT> <JSON_file>
[c] python core/collect_warp_markers.py <TRACK> <SLOT>
[d] python core/create_warp_markers.py <TRACK> <SLOT> <JSON_file>
[e] python core/move_warp_marker_relative_distance.py <TRACK> <SLOT> <MARKER> <VALUE>
[f] python core/device_snapshot.py <TRACK> <SLOT>
[g] python core/restore_device_snapshot.py <TRACK> <SLOT> <JSON_file>
[h] python core/mixer_snapshot.py <OPTION>
[i] python core/restore_mixer_snapshot.py <JSON_file> <OPTION>
[j] python core/clear_warp_markers.py <TRACK> <SLOT>
[w] python core/verify_loopback.py
[z] python core/lom_command_osc.py

extra

- midi_clip_gen
- midi_clip_op
- midi_json_gen
- midi_json_op
- midi_json_tool
- warp_clip_gen
- warp_clip_op
- warp_json_gen
- warp_json_op
- device
- mixer

python generate_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>
python generate_melody_prob.py <ROOT> <SCALE> <PROG> <INTERVAL> <PROBABILITY> <OPTION>
python generate_progression.py <ROOT> <SCALE> <PROG>
python generate_progression_with_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>

python modify_midi_prob.py <JSON> <PROBABILITY> <OPTION>
python modify_midi_timing.py <JSON> <DELAY> <OPTION1> <OPTION2>
python modify_midi_vdev.py <JSON> <VDEV_VALUE>
python arpeggiate_json.py <JSON> <INTERVAL>

python extra/midi_json_tool/midi_analyze.py <JSON_file>

python generate_grid_markers.py <TRACK> <SLOT> <STEP_LENGTH> <SWING_AMOUNT>

python randomize_device_params.py <TRACK> <DEVICE_NUMBER>

python interpolate_mixer_snapshots.py <JSON_file> <JSON_file>

Subset of the extra methods

IMPORTANT: The toolset works using OSC communication.
Therefore, the Bridge on the Assistant needs to be enabled.

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
	X		
X			
	X		
X			

```
------------------------------------------------------------------------------------------------------------
Usage: python core/clear_warp_markers.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} : None, remove all apart the first and the last markers.
{OPTION} : Use 1 for removing all warp markers.
{OPTION} : Use 2 for removing all warp markers, apart the first.
------------------------------------------------------------------------------------------------------------
Example: python core/clear_warp_markers.py 3 1
------------------------------------------------------------------------------------------------------------
This method clears warp markers from an audio clip.

------------------------------------------------------------------------------------------------------------
Usage: python core/collect_midi_clip.py <TRACK_NUMBER> <CLIP_NUMBER>
------------------------------------------------------------------------------------------------------------
<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.
------------------------------------------------------------------------------------------------------------
This method collects notes from a MIDI clip and saves them to a JSON file.


------------------------------------------------------------------------------------------------------------
Usage: python core/collect_warp_markers.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 warp markers data.
------------------------------------------------------------------------------------------------------------
Example: python core/collect_warp_markers.py 3 1
------------------------------------------------------------------------------------------------------------
This method collects warp markers from an audio clip and saves them to a JSON file.


------------------------------------------------------------------------------------------------------------
Usage: python core/create_midi_clip.py <TRACK> <SLOT> <JSON>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<JSON> : The JSON file containing the MIDI note information.
------------------------------------------------------------------------------------------------------------
Example: python core/create_midi_clip.py 1 2 data/MIDI_Clip_0_1_20260405_1343.json
------------------------------------------------------------------------------------------------------------
This method creates a MIDI clip using the note data from a JSON file.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
	X		
		X	
			X
		X	

```
------------------------------------------------------------------------------------------------------------
Usage: python core/create_warp_markers.py <TRACK_NUMBER> <CLIP_NUMBER> <JSON>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<JSON> : The JSON file containing the warp marker information.
------------------------------------------------------------------------------------------------------------
Example: python core/create_warp_markers.py 3 2 data/Warp_Markers_2_0_20260416_0927.json
------------------------------------------------------------------------------------------------------------
This method adds warp markers to an audio clip using the data from a JSON file.

------------------------------------------------------------------------------------------------------------
Usage: python core/device_snapshot.py <TRACK_NUMBER> <DEVICE_NUMBER>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<DEVICE_NUMBER> : The device number in the track (counting from left).
------------------------------------------------------------------------------------------------------------
Example: python core/device_snapshot.py 3 1
------------------------------------------------------------------------------------------------------------
This method collects a device snapshot and saves it to a JSON file.

------------------------------------------------------------------------------------------------------------
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.

------------------------------------------------------------------------------------------------------------
Usage: python core/restore_device_snapshot.py <TRACK_NUMBER> <DEVICE_NUMBER> <JSON>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<DEVICE_NUMBER> : The device number in the track (counting from left).
<JSON> : The JSON file for the snapshot to restore.
------------------------------------------------------------------------------------------------------------
Example: python core/restore_device_snapshot.py 3 1 data/Auto_Filter_snapshot_20260416_0939.json
------------------------------------------------------------------------------------------------------------
This method sends a device snapshot from a JSON file to a device.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
			X
		X	
		X	
		X	

```
------------------------------------------------------------------------------------------------------------
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 current 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.

------------------------------------------------------------------------------------------------------------
Usage: python extra/device/inspect_device_params.py <TRACK_NUMBER> <DEVICE_NUMBER>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<DEVICE_NUMBER> : The device number in the track (counting from left).
------------------------------------------------------------------------------------------------------------
Example: python extra/device/inspect_device_params.py 3 1
------------------------------------------------------------------------------------------------------------
This method shows all parameters of a device.

------------------------------------------------------------------------------------------------------------
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.

------------------------------------------------------------------------------------------------------------
Usage: python extra/device/randomize_device_params.py <TRACK> <DEVICE_NUMBER>
------------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<DEVICE_NUMBER> : The device number in the track (counting from left).
{OPTION} : Use 1 for randomizing also ON / OFF parameters.
------------------------------------------------------------------------------------------------------------
Example: python extra/device/randomize_device_params.py 3 1
------------------------------------------------------------------------------------------------------------
This method randomizes the parameters of a device.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
X			
X			
X			
X			

```
--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_clip_op/arpeggiate_clip.py <TRACK_NUMBER> <CLIP_NUMBER> <INTERVAL>
--------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<INTERVAL> : The interval in beats (1/16th = 0.25, 1/8th = 0.5, 1/4th = 1.0 etc.
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_clip_op/arpeggiate_clip.py 1 2 0.25
--------------------------------------------------------------------------------------------------------
This method retrieves notes from a MIDI clip and rebuilds the clip as an arpeggio.

--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_clip_op/list_clip_notes.py <TRACK_NUMBER> <CLIP_NUMBER>
--------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_clip_op/list_clip_notes.py 1 2
--------------------------------------------------------------------------------------------------------
This method retrieves notes from a MIDI clip and shows them as a list.

--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_clip_op/transpose_clip.py <TRACK_NUMBER> <CLIP_NUMBER> <SEMINTONES>
--------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<SEMINTONES> : The number of semitones (pitch amount) for the transposition. Positive transposes UP, negative DOWN.
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_clip_op/transpose_clip.py 1 2 -2
--------------------------------------------------------------------------------------------------------
This method transposes all notes in a MIDI clip.

--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_clip_op/transpose_conditional.py <TRACK_NUMBER> <CLIP_NUMBER> <SEMINTONES> <CONDITION>
--------------------------------------------------------------------------------------------------------
<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<SEMINTONES> : The number of semitones (pitch amount) for the transposition. Positive transposes UP, negative DOWN.
<CONDITION> : A list of notes (identified as 0=C up to 11=B) to be transposed. [0,3,11] means any note C, D# and B
will need to be transposed.
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_clip_op/transpose_conditional.py 1 2 -2 [0,3,11]
--------------------------------------------------------------------------------------------------------
This method transposes only notes in a MIDI clip which match the <CONDITION>.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
X			
X			
X			

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_gen/generate_melody.py <ROOT> <SCALE> <PROGRESSION> <INTERVAL> <RANDOMNESS>
------------------------------------------------------------------------------------------------------------
<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].
<INTERVAL> : The interval in beats (1/16th = 0.25, 1/8th = 0.5, 1/4th = 1.0 etc.
<RANDOMNESS> : The randomness from 0 to 1 to randomly mute some notes.
------------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_gen/generate_melody.py G Minor [1,5,6,4] 0.25 0.15
------------------------------------------------------------------------------------------------------------
This method creates a MIDI clip with a melody matching a given progression.

------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_gen/generate_melody_prob.py <ROOT> <SCALE> <PROG> <INTERVAL> <PROB> <OPTION>
------------------------------------------------------------------------------------------------------------
<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}.
<PROG> : The progression expressed as a list of degrees such as [1,5,6,4].
<INTERVAL> : The interval in beats (1/16th = 0.25, 1/8th = 0.5, 1/4th = 1.0 etc.
<PROB> : The probability each note will play each time, from 0 to 1.
<OPTION> : Set to 1 for having a downbeat note always play (probability 1)
------------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_gen/generate_melody_prob.py G Minor [1,5,6,4] 0.25 0.5 1
------------------------------------------------------------------------------------------------------------
This method creates a MIDI clip with a melody matching a given progression. The MIDI notes are also given a
probability.

------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_gen/generate_progression.py <ROOT> <SCALE> <PROGRESSION> {OPTION1{1,2,3}} {OPTION2}
------------------------------------------------------------------------------------------------------------
<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].
{OPTION1} : Apply the following modifiers to the progression
{OPTION1} = 1 >>>> extend all chords to 7th chords
{OPTION1} = 2 >>>> extend randomly some chords to 7th chords (using {OPTION2})
{OPTION1} = 3 >>>> spread all chords across a double octave range (no extensions)
{OPTION1} = 4 >>>> spread all chords across a double octave range (extending all chords to 7th)
{OPTION1} = 5 >>>> spread all chords across a double octave range (extending all chords to 7th using {OPTION2})
{OPTION2} : A value between 0. and 1. for probability of {OPTION1} modifiers (default 0.5, which stands for 50%)
------------------------------------------------------------------------------------------------------------
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.
------------------------------------------------------------------------------------------------------------
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
X			
X			
X			

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_gen/generate_progression_with_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>
------------------------------------------------------------------------------------------------------------
<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}.
<PROG> : The progression expressed as a list of degrees such as [1,5,6,4].
<INTERVAL> : The interval in beats (1/16th = 0.25, 1/8th = 0.5, 1/4th = 1.0 etc.
<RANDOMNESS> : The randomness from 0 to 1 to randomly mute some notes.
------------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_gen/generate_progression_with_melody.py G Minor [1,5,6,4] 0.25 0.15
------------------------------------------------------------------------------------------------------------
This method creates a MIDI clip with a given progression and a melody matching the progression.
```

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_op/arpeggiate_midi.py <JSON> <INTERVAL>
------------------------------------------------------------------------------------------------------------
<JSON> : The JSON file containing the MIDI note original information.
<INTERVAL> : The interval in beats (1/16th = 0.25, 1/8th = 0.5, 1/4th = 1.0 etc.
------------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_op/arpeggiate_midi.py data/MIDI_Clip_G_Minor_prog_20260416_1135.json 0.25
------------------------------------------------------------------------------------------------------------
This method retrieves notes from a JSON file for MIDI and rebuilds it as an arpeggio.
```

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_op/modify_midi_prob.py <JSON> <PROBABILITY> <OPTION>
------------------------------------------------------------------------------------------------------------
<JSON> : The JSON file containing the MIDI note original information.
<PROBABILITY> : The probability each note will play each time, from 0 to 1.
<OPTION> = 1 >>>> modify all notes
<OPTION> = 2 >>>> modify only notes not placed at exact beat position
<OPTION> = 3 >>>> modify only notes not placed at the beginning of a bar
------------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_op/modify_midi_prob.py data/MIDI_Clip_G_Minor_prog_20260416_1135.json 0.5 1
------------------------------------------------------------------------------------------------------------
This method retrieves notes from a JSON file for MIDI and adds a probability to notes.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
X			
X			
X			

```
--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_op/modify_midi_timing.py <JSON> <DELAY> <OPTION1> <OPTION2>
--------------------------------------------------------------------------------------------------------
<JSON> : The JSON file containing the MIDI note original information.
<DELAY> : The delay to be applied to the timing of the affected notes.
<OPTION1> = 0 >>>> modify all notes
<OPTION1> = 1 >>>> modify only notes not placed at exact beat position
<OPTION1> = 2 >>>> modify only notes not placed at the beginning of a bar
<OPTION1> = 0 >>>> modify all notes
<OPTION2> = 0 >>>> modify all notes in the same direction
<OPTION2> = 1 >>>> modify all notes in alternative direction
<OPTION2> = 2 >>>> modify all notes in random direction
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_op/modify_midi_timing.py data/MIDI_Clip_Modified_20260416_1149.json 0.25 1 2
--------------------------------------------------------------------------------------------------------
This method retrieves notes from a JSON file for MIDI and rebuilds it modifying timing of some notes.

--------------------------------------------------------------------------------------------------------
Usage: python extra/midi_json_op/modify_midi_vdev.py <JSON> <VDEV_VALUE>
--------------------------------------------------------------------------------------------------------
<JSON> : The JSON file containing the MIDI note original information.
<VDEV_VALUE> : The velocity deviation to be applied, between 0 (no deviation) and 127 (maximum variability).
--------------------------------------------------------------------------------------------------------
Example: python extra/midi_json_op/modify_midi_vdev.py data/MIDI_Clip_G_Minor_prog_20260416_1135.json 50
--------------------------------------------------------------------------------------------------------
This method retrieves notes from a JSON file for MIDI and adds velocity deviation to notes.

--------------------------------------------------------------------------------------------------------
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.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
			X
			X
			X

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/mixer/randomize_mixer_settings.py <OPTION{1,2,3}>
------------------------------------------------------------------------------------------------------------
<OPTION> = 1 >>>> randomize all tracks
<OPTION> = 2 >>>> randomize only regular tracks
<OPTION> = 3 >>>> randomize only return tracks
<OPTION> = 4 >>>> randomize only the master track
------------------------------------------------------------------------------------------------------------
Example: python extra/mixer/randomize_mixer_settings.py 1
------------------------------------------------------------------------------------------------------------
This method randomizes the mixer settings.
```

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/mixer/interpolate_mixer_snapshots.py <JSON1> <JSON2> <WEIGHT>
------------------------------------------------------------------------------------------------------------
<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/mixer/interpolate_mixer_snapshots.py data/Mixer_snapshot_20260410_1611.json
data/Mixer_snapshot_20260416_1206.json 0.33
------------------------------------------------------------------------------------------------------------
This method interpolates two mixer snapshots from JSON files and send the interpolation to the mixer.
```

```
------------------------------------------------------------------------------------------------------------
Usage: python extra/mixer/inspect_mixer_settings.py
------------------------------------------------------------------------------------------------------------
<>
------------------------------------------------------------------------------------------------------------
Example: python extra/mixer/inspect_mixer_settings.py
------------------------------------------------------------------------------------------------------------
This method shows settings in the mixer.
```

[Appendix/1]: Python methods

MIDI	Warp Markers	Device	Mixer
	X		
	X		
	X		

Usage: python extra/warp_clip_gen/generate_swing_markers.py <TRACK_NUMBER> <CLIP_NUMBER> <STEP_LENGTH> <AMOUNT>

<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<STEP_LENGTH> : the beat step for new markers. Typical value is 0.25 (1/16th reference markers).
<AMOUNT> : a float value {-1, 1}. 0.5 delays the upbeat step of 50%; -0.5 anticipates the upbeat step of 50%.

Example: python extra/warp_clip_gen/generate_swing_markers.py 3 2 0.25 0.2

This method generates new warp markers to an audio clip, eventually applying a swing through marker positioning.

Usage: python extra/warp_clip_op/move_warp_marker_relative_distance.py <TRACK_NUMBER> <CLIP_NUMBER> <PERCENTAGE>

<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<SELECTED_MARKER> : the marker to be moved. Count from left, starting with 1.
<PERCENTAGE> : a float value {0, 1} which defines how closer the marker moves to the next marker. Halfway is 0.5

Example: python extra/warp_clip_op/move_warp_marker_relative_distance.py 3 2 2 0.5

This method moves a selected warp marker of a relative distance.

Usage: python extra/warp_clip_op/remove_markers.py <TRACK_NUMBER> <CLIP_NUMBER> <LIST_MARKERS>

<TRACK_NUMBER> : The track where the target clip is located
<CLIP_NUMBER> : The slot of the target clip.
<LIST_MARKERS> : a list of markers to be removed. Count from left, starting with 1. Example: [2,6,10]

Example: python extra/warp_clip_op/remove_markers.py 3 2 [2,6,10]

This method removes selected warp markers from an audio clip.

[Appendix/1]: Python methods & supported scales

The following scripts work with scales:

```
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> <PROG> {OPTION1{1,2,3}} {OPTION2}
python extra/midi_json_gen/generate_progression_with_melody.py <ROOT> <SCALE> <PROG> <INTERVAL> <RANDOMNESS>
```

The following scripts recognize scales:

```
python extra/midi_json_tool/midi_analyze.py <JSON>
```