Ableton Live 11 Complete API Reference & Installation Guide

Table of Contents

Overview

Ableton Live 11 provides multiple ways to programmatically control the DAW:

1. **AbletonOSC** - OSC (Open Sound Control) interface via MIDI Remote Script
2. **Python Remote Scripts** - Native Python API for control surfaces
3. **Max for Live (M4L)** - Visual programming with Live API access
4. **Live Object Model (LOM)** - The underlying object hierarchy

Key Facts

- **Python Version**: Live 11 uses **Python 3** (Live 10 and earlier used Python 2.7)
- **AbletonOSC Ports**: Listens on port **11000**, sends replies on port **11001**
- **API Hierarchy**: Application → Song → Tracks → Clips/Devices → Parameters

---

Installation Paths & File Locations

Application Installation

Windows

```
C:\ProgramData\Ableton\Live 11.x.x\
├── Program\
│   └── Ableton Live 11 Suite.exe
├── Resources\
│   ├── MIDI Remote Scripts\
│   │   ├── _Framework\
│   │   ├── _Generic\
│   │   ├── _Tools\
│   │   ├── ableton\
│   │   │   └── v2\
│   │   ├── APC40\
│   │   ├── Push2\
│   │   └── [other control surfaces]\
│   ├── Core Library\
│   └── Python\
└── ...
```

macOS

```
/Applications/Ableton Live 11 Suite.app/
└── Contents/
    └── App-Resources/
        ├── MIDI Remote Scripts/
        │   ├── _Framework/
        │   ├── _Generic/
        │   ├── _Tools/
        │   ├── ableton/
        │   │   └── v2/
        │   └── [control surfaces]/
        ├── Core Library/
        └── Python/
```

User Data Locations

Windows

```
C:\Users\[username]\
├── Documents\
│   └── Ableton\
│       ├── User Library\
│       │   ├── Remote Scripts\       # Custom MIDI scripts (Live 10.1.13+)
│       │   ├── Presets\
│       │   ├── Samples\
│       │   ├── Templates\
│       │   ├── Clips\
│       │   ├── Defaults\
│       │   └── Grooves\
│       └── Factory Packs\
├── AppData\
│   └── Roaming\
│       └── Ableton\
│           └── Live 11.x.x\
│               └── Preferences\
│                   ├── Template.als
│                   ├── Options.txt
│                   └── Log.txt       # Debug/error log file
│
```

macOS

```
/Users/[username]/
├── Music/
│   └── Ableton/
│       ├── User Library/
│       │   ├── Remote Scripts/
│       │   ├── Presets/
│       │   ├── Samples/
│       │   ├── Templates/
│       │   ├── Clips/
│       │   ├── Defaults/
│       │   └── Grooves/
│       └── Factory Packs/
└── Library/
    ├── Preferences/
    │   └── Ableton/
    │       └── Live 11.x.x/
    │           ├── Log.txt
    │           └── Options.txt
    └── Application Support/
        └── Ableton/
```

Core Library Location

- **Windows**: `C:\ProgramData\Ableton\Live\Resources\Core Library`
- **macOS**: `/Applications/Ableton Live 11 Suite.app/Contents/App-Resources/Core Library`

Third-Party Remote Scripts Installation

As of Live 10.1.13+, place custom scripts in:

- **Windows**: `\Users\[username]\Documents\Ableton\User Library\Remote Scripts\`
- **macOS**: `/Users/[username]/Music/Ableton/User Library/Remote Scripts/`

Live Object Model (LOM) Architecture

The Live Object Model is a hierarchical structure representing all accessible parts of Live.

Root Objects

```
live_app          → Application object
live_set          → Current Song/Set
```

```
control_surfaces N    → Control surface at index N
this_device           → M4L device containing the path
```

Object Hierarchy

```
Application (live_app)
├── view
├── control_surfaces[]
└── document → Song

Song (live_set)
├── view
├── master_track
├── tracks[]
│   ├── view
│   ├── mixer_device
│   │   ├── volume
│   │   ├── panning
│   │   └── sends[]
│   ├── devices[]
│   │   ├── parameters[]
│   │   └── view
│   ├── clip_slots[]
│   │   └── clip
│   │       ├── view
│   │       └── notes (MIDI clips)
│   └── arrangement_clips[]
├── return_tracks[]
├── scenes[]
├── cue_points[]
├── visible_tracks[]
└── groove_pool
```

Core LOM Classes

Application

```
python
```

```
class Application:
    # Properties
    browser                    # Browser object
    control_surfaces           # List of control surfaces
    current_dialog_button_count  # int
    current_dialog_message       # str
    view                       # Application.View

    # Methods
    get_bugfix_version()       # Returns: int (e.g., 2 in 11.0.2)
    get_document()             # Returns: Song
    get_major_version()        # Returns: int (e.g., 11)
    get_minor_version()        # Returns: int (e.g., 0)
    press_current_dialog_button(index)
```

Song

```
python
```

```
class Song:
    # Transport Properties
    is_playing                  # bool
    current_song_time           # float (beats)
    tempo                       # float (BPM, 20-999)
    signature_numerator         # int
    signature_denominator       # int

    # Loop Properties
    loop                        # bool
    loop_start                  # float (beats)
    loop_length                 # float (beats)

    # Record Properties
    record_mode                 # bool
    session_record              # bool
    arrangement_overdub         # bool
    punch_in                    # bool
    punch_out                   # bool
    metronome                   # bool

    # Track Collections
    tracks                      # list of Track
    return_tracks               # list of Track
    master_track                # Track
    visible_tracks              # list of Track
    scenes                      # list of Scene

    # Quantization
    clip_trigger_quantization   # int (0=None, 1=8bars...14=1/32)
    midi_recording_quantization  # int

    # Methods
    start_playing()
    stop_playing()
    continue_playing()
    stop_all_clips()
    tap_tempo()
    trigger_session_record()
    undo()
    redo()
    capture_midi()
    create_audio_track(index)
```

```
create_midi_track(index)
create_return_track()
create_scene(index)
delete_track(index)
delete_return_track(index)
delete_scene(index)
duplicate_track(index)
duplicate_scene(index)
jump_by(beats)
jump_to_next_cue()
jump_to_prev_cue()
```

Track

python

```
class Track:
    # Identity
    name                    # str
    color                   # int (RGB)
    color_index             # int (0-69)

    # Mixer
    volume                  # via mixer_device
    panning                 # via mixer_device

    # States
    arm                     # bool (record arm)
    mute                    # bool
    solo                    # bool
    current_monitoring_state     # int (0=In, 1=Auto, 2=Off)

    # Routing
    available_input_routing_channels
    available_input_routing_types
    available_output_routing_channels
    available_output_routing_types
    input_routing_channel
    input_routing_type
    output_routing_channel
    output_routing_type

    # Structure
    can_be_armed            # bool
    has_audio_input         # bool
    has_audio_output        # bool
    has_midi_input          # bool
    has_midi_output         # bool
    is_foldable             # bool (is group track)
    is_grouped              # bool
    is_visible              # bool
    fold_state              # bool (group folded)

    # Children
    clip_slots              # list of ClipSlot
    devices                 # list of Device
    mixer_device            # MixerDevice
    arrangement_clips       # list of Clip
    # Playback
    playing_slot_index          # int (-2=stop, -1=none, 0+=slot)
    fired_slot_index            # int

    # Meters
    output_meter_left           # float (0.0-1.0)
    output_meter_right          # float (0.0-1.0)
    output_meter_level          # float (0.0-1.0)

    # Methods
    stop_all_clips()
```

ClipSlot

python

```python
class ClipSlot:
    # Properties
    has_clip                    # bool
    clip                        # Clip or None
    has_stop_button             # bool
    is_playing                  # bool
    is_recording                # bool
    is_triggered                # bool

    # Methods
    fire()
    stop()
    create_clip(length)         # Creates empty MIDI clip
    delete_clip()
    duplicate_clip_to(target_slot)
```

Clip

python

```
class Clip:
    # Identity
    name                    # str
    color                   # int
    color_index             # int

    # Type
    is_audio_clip           # bool
    is_midi_clip            # bool

    # Playback
    is_playing              # bool
    is_recording            # bool
    is_overdubbing          # bool
    is_triggered            # bool
    playing_position        # float (beats)
    will_record_on_start    # bool

    # Length/Position
    length                  # float (beats)
    start_time              # float (arrangement position)
    end_time                # float
    loop_start              # float
    loop_end                # float
    start_marker            # float
    end_marker              # float
    position                # float (loop start alias)

    # Audio Properties
    gain                    # float (0.0-1.0)
    pitch_coarse            # int (semitones, -48 to +48)
    pitch_fine              # float (cents, -50 to +50)
    warping                 # bool
    warp_mode               # int (0=Beats,1=Tones,2=Texture,3=Re-Pitch,4=Complex,6=Pro)
    sample_length           # float (for audio clips)
    file_path               # str (for audio clips)

    # MIDI Properties
    muted                   # bool

    # Launch
    launch_mode             # int (0=Trigger,1=Gate,2=Toggle,3=Repeat)
    launch_quantization     # int (0=Global,1=None,2=8Bars...14=1/32)
```

```
legato                    # bool
velocity_amount           # float (0.0-1.0)
ram_mode                  # bool (load into RAM)
has_groove                # bool

# Methods
fire()
stop()
duplicate_loop()

# MIDI Note Methods (MIDI clips only)
get_notes(start_time, time_span, start_pitch, pitch_span)
get_notes_extended(from_time, time_span, from_pitch, pitch_span)
set_notes(notes_tuple)
add_new_notes(notes_tuple)
replace_selected_notes(notes_tuple)
remove_notes(start_time, time_span, start_pitch, pitch_span)
remove_notes_extended(from_time, time_span, from_pitch, pitch_span)
select_all_notes()
deselect_all_notes()
get_selected_notes()

# Audio Warp Methods
add_warp_marker(beat_time, sample_time)
remove_warp_marker(beat_time)
move_warp_marker(beat_time, new_beat_time)
```

Device

python

```
class Device:
    # Identity
    name                    # str
    class_name              # str (e.g., "Operator", "Reverb", "PluginDevice")
    type                    # DeviceType (1=audio_effect, 2=instrument, 4=midi_effect)

    # State
    is_active               # bool

    # Parameters
    parameters              # list of DeviceParameter

    # Methods
    store_chosen_bank(bank_index, parameter_indices)
```

DeviceParameter

python

```
class DeviceParameter:
    # Identity
    name                    # str
    original_name           # str

    # Value
    value                   # float
    default_value           # float
    min                     # float
    max                     # float

    # Type
    is_enabled              # bool
    is_quantized            # bool (discrete values only)
    value_items             # list of str (for quantized params)

    # Automation
    automation_state        # AutomationState
    state                   # ParameterState
```

Scene

python

```
class Scene:
    # Identity
    name                    # str
    color                   # int
    color_index             # int

    # State
    is_empty                # bool
    is_triggered            # bool

    # Tempo
    tempo                   # float
    tempo_enabled           # bool

    # Time Signature
    time_signature_numerator     # int
    time_signature_denominator   # int
    time_signature_enabled       # bool

    # Children
    clip_slots              # list of ClipSlot

    # Methods
    fire()
    fire_as_selected()
    set_fire_button_state(state)
```

AbletonOSC API Reference

AbletonOSC is the recommended way to control Live 11 via OSC. Install by copying the `AbletonOSC` folder to your Remote Scripts directory.

Connection Settings

- **Listen Port**: 11000
- **Reply Port**: 11001
- **Protocol**: UDP

Application API

Address	Params	Response	Description
`/live/test`		'ok'	Test connection
`/live/application/get/version`		major, minor	Get Live version
`/live/api/reload`			Reload AbletonOSC
`/live/api/get/log_level`		level	Get log level
`/live/api/set/log_level`	level		Set log level (debug/info/warning/error/critical)
`/live/api/show_message`	message		Show message in status bar

Song API

Transport Methods

Address	Params	Description
`/live/song/start_playing`		Start playback
`/live/song/stop_playing`		Stop playback
`/live/song/continue_playing`		Resume playback
`/live/song/stop_all_clips`		Stop all clips
`/live/song/tap_tempo`		Tap tempo
`/live/song/trigger_session_record`		Toggle session record
`/live/song/undo`		Undo
`/live/song/redo`		Redo
`/live/song/capture_midi`		Capture MIDI
`/live/song/jump_by`	beats	Jump by beats
`/live/song/jump_to_next_cue`		Next cue point
`/live/song/jump_to_prev_cue`		Previous cue point

Track/Scene Creation

Address	Params	Description
`/live/song/create_audio_track`	index	Create audio track (-1=end)
`/live/song/create_midi_track`	index	Create MIDI track (-1=end)
`/live/song/create_return_track`		Create return track
`/live/song/create_scene`	index	Create scene (-1=end)
`/live/song/delete_track`	index	Delete track
`/live/song/delete_return_track`	index	Delete return track
`/live/song/delete_scene`	index	Delete scene
`/live/song/duplicate_track`	index	Duplicate track
`/live/song/duplicate_scene`	index	Duplicate scene

Song Properties (Get/Set)

Property	Type	Description
`tempo`	float	BPM (20-999)
`metronome`	bool	Metronome on/off
`is_playing`	bool	Playing state
`current_song_time`	float	Position in beats
`loop`	bool	Loop enabled
`loop_start`	float	Loop start (beats)
`loop_length`	float	Loop length (beats)
`record_mode`	bool	Record mode
`session_record`	bool	Session record
`arrangement_overdub`	bool	Arrangement overdub
`punch_in`	bool	Punch in
`punch_out`	bool	Punch out
`signature_numerator`	int	Time sig numerator
`signature_denominator`	int	Time sig denominator
`clip_trigger_quantization`	int	Clip trigger quantize
`midi_recording_quantization`	int	MIDI record quantize
`groove_amount`	float	Global groove amount
`root_note`	int	Root note
`scale_name`	str	Scale name

Usage: `/live/song/get/tempo`, `/live/song/set/tempo 120.0`

Bulk Data Query

```
/live/song/get/track_data 0 12 track.name clip.name clip.length
```

Returns: `[track_0_name, clip_0_0_name, clip_0_0_length, clip_0_1_name, ...]`

Track API

Track Properties (Get/Set)

Property	Type	Description
`name`	str	Track name
`color`	int	RGB color
`color_index`	int	Color index (0-69)
`arm`	bool	Record arm
`mute`	bool	Mute
`solo`	bool	Solo
`volume`	float	Volume (0.0-1.0)
`panning`	float	Pan (-1.0 to 1.0)
`send`	float	Send level (needs send_id)
`current_monitoring_state`	int	0=In, 1=Auto, 2=Off
`fold_state`	bool	Group folded
`playing_slot_index`	int	Currently playing slot
`fired_slot_index`	int	Triggered slot

Usage: `/live/track/get/volume 0` → returns `0, 0.85`

Track Routing

Property	Description
`available_input_routing_channels`	List input channels
`available_input_routing_types`	List input types
`available_output_routing_channels`	List output channels
`available_output_routing_types`	List output types
`input_routing_channel`	Current input channel
`input_routing_type`	Current input type
`output_routing_channel`	Current output channel
`output_routing_type`	Current output type

Track Methods

Address	Params	Description
`/live/track/stop_all_clips`	track_id	Stop all clips on track

Clip Slot API

Address	Params	Description
`/live/clip_slot/fire`	track_id, slot_id	Fire clip slot
`/live/clip_slot/create_clip`	track_id, slot_id, length	Create empty MIDI clip
`/live/clip_slot/delete_clip`	track_id, slot_id	Delete clip
`/live/clip_slot/get/has_clip`	track_id, slot_id	Check if slot has clip
`/live/clip_slot/get/has_stop_button`	track_id, slot_id	Check stop button
`/live/clip_slot/set/has_stop_button`	track_id, slot_id, state	Set stop button
`/live/clip_slot/duplicate_clip_to`	t_id, s_id, target_t, target_s	Duplicate clip

Clip API

Clip Control

Address	Params	Description
`/live/clip/fire`	track_id, clip_id	Fire clip
`/live/clip/stop`	track_id, clip_id	Stop clip
`/live/clip/duplicate_loop`	track_id, clip_id	Duplicate loop

Clip Properties (Get/Set)

Property	Type	Description
`name`	str	Clip name
`color`	int	RGB color
`color_index`	int	Color index (0-69)
`gain`	float	Clip gain
`pitch_coarse`	int	Semitones (-48 to +48)
`pitch_fine`	float	Cents (-50 to +50)
`loop_start`	float	Loop start (beats)
`loop_end`	float	Loop end (beats)
`start_marker`	float	Start marker
`end_marker`	float	End marker
`position`	float	Loop position
`warping`	bool	Warp enabled
`warp_mode`	int	0=Beats,1=Tones,2=Texture,3=Re-Pitch,4=Complex,6=Pro
`launch_mode`	int	0=Trigger,1=Gate,2=Toggle,3=Repeat
`launch_quantization`	int	0=Global,1=None,2=8Bars...14=1/32
`muted`	bool	Clip muted
`legato`	bool	Legato mode
`velocity_amount`	float	Velocity amount (0.0-1.0)
`ram_mode`	bool	Load to RAM

Clip Read-Only Properties

Property	Description
`length`	Clip length in beats
`sample_length`	Audio sample length
`start_time`	Arrangement start time
`file_path`	Audio file path
`is_audio_clip`	Is audio clip
`is_midi_clip`	Is MIDI clip
`is_playing`	Is playing
`is_recording`	Is recording
`is_overdubbing`	Is overdubbing
`playing_position`	Current play position
`has_groove`	Has groove

MIDI Note Operations

```
# Get notes
/live/clip/get/notes track_id clip_id [start_pitch pitch_span start_time time_span]
→ Returns: track_id, clip_id, pitch, start_time, duration, velocity, mute, ...

# Add notes
/live/clip/add/notes track_id clip_id pitch start_time duration velocity mute ...

# Remove notes
/live/clip/remove/notes track_id clip_id [start_pitch pitch_span start_time time_span]
```

Listening to Playing Position

```
/live/clip/start_listen/playing_position track_id clip_id
→ Continuously sends: /live/clip/get/playing_position track_id clip_id position

/live/clip/stop_listen/playing_position track_id clip_id
```

Scene API

Scene Methods

Address	Params	Description
`/live/scene/fire`	scene_id	Fire scene
`/live/scene/fire_as_selected`	scene_id	Fire and select next
`/live/scene/fire_selected`		Fire selected scene

Scene Properties

Property	Type	Description
`name`	str	Scene name
`color`	int	RGB color
`color_index`	int	Color index
`tempo`	float	Scene tempo
`tempo_enabled`	bool	Tempo enabled
`time_signature_numerator`	int	Time sig numerator
`time_signature_denominator`	int	Time sig denominator
`time_signature_enabled`	bool	Time sig enabled
`is_empty`	bool	Scene is empty
`is_triggered`	bool	Scene triggered

Device API

Address	Params	Response	Description
`/live/device/get/name`	t_id, d_id	t_id, d_id, name	Device name
`/live/device/get/class_name`	t_id, d_id	t_id, d_id, class	Device class

Address	Params	Response	Description
/live/device/get/type	t_id, d_id	t_id, d_id, type	1=audio_effect, 2=instrument, 4=midi_effect
/live/device/get/num_parameters	t_id, d_id	t_id, d_id, count	Parameter count
/live/device/get/parameters/name	t_id, d_id	t_id, d_id, names...	All param names
/live/device/get/parameters/value	t_id, d_id	t_id, d_id, values...	All param values
/live/device/get/parameters/min	t_id, d_id	t_id, d_id, mins...	All param minimums
/live/device/get/parameters/max	t_id, d_id	t_id, d_id, maxs...	All param maximums
/live/device/set/parameters/value	t_id, d_id, vals...		Set all params
/live/device/get/parameter/value	t_id, d_id, p_id	t_id, d_id, p_id, val	Single param value
/live/device/set/parameter/value	t_id, d_id, p_id, val		Set single param
/live/device/get/parameter/value_string	t_id, d_id, p_id	t_id, d_id, p_id, str	Formatted value

View API

Address	Params	Response	Description
/live/view/get/selected_track		track_id	Selected track
/live/view/set/selected_track	track_id		Select track
/live/view/get/selected_scene		scene_id	Selected scene
/live/view/set/selected_scene	scene_id		Select scene
/live/view/get/selected_clip		track_id, scene_id	Selected clip

Address	Params	Response	Description
/live/view/set/selected_clip	track_id, scene_id		Select clip
/live/view/get/selected_device		track_id, device_id	Selected device
/live/view/set/selected_device	track_id, device_id		Select device

MIDI Map API

Address	Params	Description
/live/midimap/map_cc	track_id, device_id, param_id, channel, cc	Map CC to parameter

Note: MIDI channels are zero-indexed (channel 1 = index 0)

Listener Pattern

For any property, you can listen for changes:

```
/live/[object]/start_listen/[property] [ids...]
→ Sends updates to: /live/[object]/get/[property]

/live/[object]/stop_listen/[property] [ids...]
```

Example:

```
/live/song/start_listen/tempo
→ Whenever tempo changes, receives: /live/song/get/tempo 120.0

/live/song/start_listen/beat
→ On each beat, receives: /live/song/get/beat beat_number
```

Python Remote Scripts API

Script Structure

A minimal Remote Script requires:

```
YourScript/
├── __init__.py
└── YourScript.py
```

__init__.py

```python
from .YourScript import YourScript


def create_instance(c_instance):
    return YourScript(c_instance)
```

YourScript.py (using _Framework)

```python
```

```
from _Framework.ControlSurface import ControlSurface
from _Framework.TransportComponent import TransportComponent
from _Framework.ButtonElement import ButtonElement
from _Framework.EncoderElement import EncoderElement

# MIDI Types
MIDI_NOTE_TYPE = 0
MIDI_CC_TYPE = 1

class YourScript(ControlSurface):
    def __init__(self, c_instance):
        ControlSurface.__init__(self, c_instance)
        with self.component_guard():
            self._setup_transport()
            self._setup_mixer()

    def _setup_transport(self):
        transport = TransportComponent()
        # ButtonElement(is_momentary, msg_type, channel, identifier)
        transport.set_play_button(ButtonElement(True, MIDI_NOTE_TYPE, 0, 60))
        transport.set_stop_button(ButtonElement(True, MIDI_NOTE_TYPE, 0, 61))
        transport.set_record_button(ButtonElement(True, MIDI_NOTE_TYPE, 0, 62))

    def song(self):
        return self._c_instance.song()

    def log_message(self, message):
        # Writes to Live's Log.txt
        ControlSurface.log_message(self, message)

    def disconnect(self):
        ControlSurface.disconnect(self)
```

Live Module Access

```
python
```

```python
import Live

# Access application
app = Live.Application.get_application()
version = app.get_major_version()

# Access song (from within ControlSurface)
song = self.song()
tempo = song.tempo
song.tempo = 120.0

# Access tracks
for track in song.tracks:
    print(track.name)
    track.mute = False

# Access clips
track = song.tracks[0]
clip_slot = track.clip_slots[0]
if clip_slot.has_clip:
    clip = clip_slot.clip
    clip.fire()

# Access devices
for device in track.devices:
    print(f"{device.name}: {device.class_name}")
    for param in device.parameters:
        print(f"  {param.name}: {param.value}")
```

Listener Pattern (Python)

python

```
def setup_listeners(self):
    self.song().add_tempo_listener(self._on_tempo_changed)
    self.song().add_is_playing_listener(self._on_playing_changed)

def _on_tempo_changed(self):
    self.log_message(f"Tempo changed to {self.song().tempo}")

def _on_playing_changed(self):
    if self.song().is_playing:
        self.log_message("Playback started")

def disconnect(self):
    self.song().remove_tempo_listener(self._on_tempo_changed)
    self.song().remove_is_playing_listener(self._on_playing_changed)
    ControlSurface.disconnect(self)
```

Framework Classes

The `_Framework` folder contains utility classes for building control surfaces:

Core Classes

Class	Purpose
`ControlSurface`	Base class for all scripts
`ControlSurfaceComponent`	Base for components
`ControlElement`	Base for MIDI elements
`ButtonElement`	MIDI note/CC button
`EncoderElement`	Rotary encoder
`SliderElement`	Fader/slider
`ButtonMatrixElement`	Grid of buttons

Components

Class	Purpose
`TransportComponent`	Play/stop/record controls
`MixerComponent`	Volume/pan/sends
`SessionComponent`	Clip launching grid
`DeviceComponent`	Device parameter control
`ChannelStripComponent`	Single channel strip
`ClipSlotComponent`	Single clip slot

Modern v2 Classes (Live 10+)

Located in `ableton/v2/`:

```
ableton/
└── v2/
    ├── control_surface/
    │   ├── control_surface.py
    │   ├── component.py
    │   └── elements.py
    └── base/
        └── ...
```

File Types & Extensions

Extension	Description
`.als`	Ableton Live Set (project file)
`.alc`	Ableton Live Clip
`.adg`	Ableton Device Group (rack preset)
`.adv`	Ableton Device Preset
`.agr`	Ableton Groove

Extension	Description
`.asd`	Ableton Sample Analysis Data
`.ask`	Ableton Skin (UI theme)
`.ams`	Ableton Modulation Set (Operator)
`.amxd`	Max for Live Device
`.alp`	Ableton Live Pack
`.py`	Python source (Remote Scripts)
`.pyc`	Python compiled bytecode

Resources & Documentation Links

Official Documentation

- Live Object Model (Cycling '74)
- Live API Overview
- Installing Remote Scripts

API Documentation

- Live 11.0 Python API XML
- NSUSpray Live API Docs
- Structure Void MIDI Remote Scripts

AbletonOSC

- AbletonOSC GitHub
- PyLive (Python client)

Decompiled Scripts

- Live 11 Remote Scripts (cylab)
- Live 10.1 Remote Scripts

Tutorials

- Framework Classes Introduction
- Remote Scripts Blog

Tools

- Control Surface Studio - Visual script builder
- MIDI Monitor (macOS)

Quick Reference: Common Operations

Start/Stop Playback (OSC)

```
/live/song/start_playing
/live/song/stop_playing
/live/song/continue_playing
```

Fire Clip (OSC)

```
/live/clip/fire 0 0      # Track 0, Clip 0
```

Change Tempo (OSC)

```
/live/song/set/tempo 128.0
```

Get All Track Names (OSC)

```
/live/song/get/track_names
```

Arm Track for Recording (OSC)

```
/live/track/set/arm 0 1   # Track 0, armed
```

Set Device Parameter (OSC)

```
/live/device/set/parameter/value 0 0 1 0.5  # Track 0, Device 0, Param 1, Value 0.5
```

Create MIDI Clip (OSC)

```
/live/clip_slot/create_clip 0 0 4.0  # Track 0, Slot 0, 4 beats long
```

Add MIDI Notes (OSC)

```
/live/clip/add/notes 0 0 60 0.0 0.5 100 0  # C4 at beat 0, 0.5 beat duration, velocity 100
```

Document compiled for JarvisAbleton AI Assistant Last updated: January 2026