

Scripts and Extensions for Hope Creative

A guide for automating Hope Creative with JavaScript and for building extensions, dockable panels with their own UI that ship as signed `.hopex` packages.

Scripting API 1 · Host version 1.0 (pre-release) · Windows x64 (macOS planned) · October 2026

Contents

- | | |
|--|------------------------------------|
| 1. Scripts and extensions | 9. Elements and drawing |
| 2. Running a script | 10. Undo, events and storage |
| 3. A first script | 11. Timers, commands and dialogs |
| 4. The objects | 12. Developer Mode |
| 5. Values and finding properties | 13. Packaging and signing (.hopex) |
| 6. Modifiers, Text Animator, Color Corrector | 14. Licenses for paid extensions |
| 7. The action journal | 15. Examples |
| 8. Extension panels | |

1. Scripts and extensions

- **Scripts** are JavaScript files that control Hope Creative: they build comps, add layers and effects, set keyframes and eases, render frames and start exports.
- **Extensions** are panels with their own UI that dock beside the app's own panels. They are built from the same API plus Hope's UI elements, so they follow Dark / Light and the accent colour on their own.

Both run in QuickJS inside Hope Creative (no browser, no Node.js). For pictures and values over time — effects and modifiers — see the *Native Plug-in Guide*.

2. Running a script

- **Tools** ▶ **Scripts** lists every `.js` file in `Documents\Hope Creative\Scripts`.
- **Tools** ▶ **Scripts** ▶ **Run Script File...** runs any other script.
- **Tools** ▶ **Scripts** ▶ **Open Scripts Folder** (also **Developer** ▶ **Open Scripts Folder**) opens the folder. Hope Creative keeps `hope.d.ts` and a `jsconfig.json` in it, so VS Code autocompletes the whole API, including every effect name.

Each script run is **one undo step** ("Script: <name>"). If a script fails, the status bar shows the error and its line. Whatever the script did up to that line stays in the project and can be undone in one step. A script that runs longer than 60 seconds is stopped. `console.log()` and `hope.log()` write to the session log.

3. A first script

```
const comp = hope.project.createComp({ name: "Promo", width: 1920, height: 1080, fps: 25, duration: 6
});
comp.open();

const bg = comp.addSolid({ name: "Background", color: "#101418" });
const title = comp.addText("Hello", { size: 140 });

// keys with eases: x = share of the time (0..1), y = share of the change
title.property("Position").setKeys([
  { time: 0, value: [-400, 540], interpolation: "bezier", easeOut: [0.3, 0] },
  { time: 1.2, value: [960, 540], easeIn: [0.2, 1] },
]);
title.property("Opacity").setValue(90);

bg.addEffect("Gaussian Blur").property("Blurriness").setValue(8);
comp.renderFrame("C:/Temp/promo_frame.png", 1.0);
```

4. The objects

Object	What it offers
hope	project, activeComp, effects (every effect, what it's for, its parameters), modifierTypes, textAnimatorInfo, looks, layerSets, aiStatus, log(), alert(), undoGroup(name, fn), undo(), redo(), file.read/write/exists
hope.project	items, item(name or id), comps, comp(name), createComp({...}), createFolder(), importFile(path), save(), open(path), missingFootage
Item	name, type, comp, remove(); footage: path, missing, replace(path) (Replace Footage), relink(path) (a moved file; other missing files found the same way come along), proxy, proxyState, createProxy() (made in the background), removeProxy()
Comp	layers, layer(name, index or {id}), selectedLayers, selectedKeys (the Timeline's selected keyframes, also settable: [{property, times}]), time (the playhead), add...() for every layer type, markers, addMarker(), setWorkArea(), setSettings(), precompose(), renderFrame(), export({format, folder, name, motionBlur}), open(), insertLayerSet() / saveLayerSet(), bakePhysics(), applyLook()
Layer	name, index, moveTo(), inPoint / outPoint, shift(), visible, threeD, blendMode, parent, matte / matteMode / setMatte(layer, mode) (track matte: "Alpha", "Alpha Inverted", "Luma", "Luma Inverted"), duplicate(), split(), remove(), properties, property(name), effects, effect(name), addEffect(name), text animator, text styles, grade, looks, physics, generateCaptions(), autoCut()

Object	What it offers
Effect	<code>name</code> , <code>enabled</code> , <code>moveTo()</code> , <code>properties</code> , <code>property(name)</code> , <code>remove()</code> ; Rigging: <code>addPin()</code> , <code>removePin()</code> ; AI Mask: <code>maskPoints()</code> , <code>trackMask()</code>
Property	<code>value</code> (at the playhead), <code>valueAt(t)</code> , <code>setValue(v, t)</code> , <code>keys</code> , <code>setKey()</code> , <code>setKeys()</code> , <code>removeKey()</code> , <code>setAnimated()</code> , <code>expression</code> , <code>reset()</code> , <code>modifiers</code> , <code>addModifier(type)</code>
Modifier	<code>name</code> , <code>type</code> , <code>enabled</code> , <code>settings</code> , <code>setting(name)</code> (a Property), <code>remove()</code>

5. Values and finding properties

- Times are in **seconds**. Numbers are numbers. Points are `[x, y]` in pixels.
- Colours are `[r, g, b, a]` (0..1) or `"#rrggbb"`.
- Menus take an entry's number or its name (`layer.property("Shape").setValue("Sphere")`).
- Every layer property can be found by the name it shows (`"Position"`) or by its match name (`"transform.position"`); the same goes for effect parameters (`"Blurriness"`). `layer.properties` and `hope.effects` list them all.

6. Modifiers, Text Animator, Color Corrector

```
const t = hope.activeComp.addText("Hello World", {size: 90});
t.property("Position").addModifier("Shake").setting("Amplitude").setValue(0.3);
t.applyTextAnimatorPreset("in", "Fade In Words Up");
t.setTextAnimator({in: {follower: "Word", params: {opacity: {on: true, curve: "Ease Out"}}}}); //
only what changes
t.setTextStyle({color: "#ff7a30"}, {find: "World"});
hope.activeComp.addAdjustment().setGrade({exposure: 0.3, lift: {x: 0.1}});
```

`hope.textAnimatorInfo` lists followers, parameters, curve and preset names; `hope.modifierTypes` the modifiers and their settings. The AI tools (`generateCaptions`, `autoCut`, `maskPoints`, `trackMask`) run in the app's background; `hope.aiStatus` says how far they are.

7. The action journal

Hope Creative writes every undo step of a session as a script into its log folder (**Developer ▶ Show Log Files**, `journal-<time>.js`), with a snapshot of the project the session started from beside it. Running the journal opens that snapshot and repeats the session step by step, ending with exactly the same project. It's also a good way to learn the API: do something by hand, then read how the journal wrote it.

8. Extension panels

An extension is a folder with a `manifest.json` and a script. During development it lives in `Documents\Hope Creative\Extensions\<name>` (**Developer ▶ Extension Folders ▶ In Development**) and loads in Developer Mode.

```
{ "id": "com.example.easer", "name": "Easer", "version": "1.0.0", "main": "main.js", "panel": {  
  "width": 320 } }
```

Developer ▶ Extensions lists every extension and shows or hides its panel (extensions live in the Custom workspace; choosing one places it where you click). The panel's script keeps running as long as Hope Creative is open, so variables keep their values.

`hope.ui.panel(render)` takes a function that returns the element tree. After every callback, and whenever the project, the selection or the comp changes, the panel calls it again and redraws from the result.

```
const ui = hope.ui;  
let amount = hope.storage.get("amount", 50); // kept between sessions  
  
ui.panel(() => ui.column({ gap: 8 }, [  
  ui.label("Easer", { font: "title" }),  
  ui.slider({ label: "Amount", value: amount, min: 0, max: 100, decimals: 0,  
    onChange: (v) => { amount = v; hope.storage.set("amount", v); } }),  
  ui.canvas({ height: 120, draw: (g, w, h) => {  
    g.fillRect(0, 0, w, h, "raised", 10);  
    g.beginPath(); g.moveTo(10, h - 10); g.cubicTo(w * 0.4, h - 10, w * 0.6, 10, w - 10, 10);  
    g.stroke("accent", 2);  
  } })),  
  ui.button("Apply", () => {  
    for (const layer of hope.activeComp.selectedLayers) layer.property("Opacity").setValue(amount);  
  }, { style: "primary" })),  
]));  
  
hope.on("selectionChanged", () => ui.refresh());  
ui.menu(() => [{ label: "Reset", onClick: () => { amount = 50; } }]); // the panel's right-click  
menu
```

9. Elements and drawing

- **Layout:** `column`, `row` (`grow` shares the leftover width), `group` (collapsible, like Effect Controls), `separator`, `spacer`.
- **Controls:** `label`, `button`, `checkbox`, `slider`, `number` (drag to scrub, double-click to type), `dropdown`, `text`, `color`, `tabs`, `list`, `progress`.
- **Ease editor:** `curve` (`value: [x1, y1, x2, y2]`), handles, grid, overshoot room, Shift snaps; a drag is one undo step).
- **Drawing:** `canvas`, with `g.fillRect`, `g.line`, `g.circle`, `paths` (`beginPath` / `moveTo` / `cubicTo` / `fill` / `stroke`) and `g.text`. It reports `onMouseDown` / `onMouseMove` / `onMouseUp` / `onWheel` with coordinates on the canvas.
- **More on the canvas:**
 - **Gradients:** `g.linearGradient(x0, y0, x1, y1, stops)` and `g.radialGradient(cx, cy, r, stops)` go wherever a colour goes: fills, strokes, text. `stops` is a list of colours spread evenly, or `[[0, "#000"], [1, "accent"]]`.
 - **Transforms and clipping:** `save` / `restore`, `translate`, `rotate(degrees, cx, cy)`, `scale`, `clipRect(x, y, w, h, radius)`, and `clip()` to the current path.
 - **Looks:** `setOpacity(a)`, `setBlend("multiply" | "screen" | "add" ...)`, `setDash([4, 3])`, `setShadow(color, blur, dx, dy)`. `save` / `restore` keep them too.

- **Text** takes `size` for any point size.
- **Keys:** `onKeyDown` / `onKeyUp` get `{key, shift, ctrl, alt, repeat}` once the canvas was clicked, while the mouse is over the panel. Return `true` for a key you used; every other key still reaches the app's shortcuts.

```
let angle = 0;
ui.canvas({ height: 140, draw: (g, w, h) => {
  g.fillRect(0, 0, w, h, g.linearGradient(0, 0, w, 0, ["#1b2a4a", "accent"]), 10);
  g.save();
  g.rotate(angle, w / 2, h / 2);
  g.setShadow("#00000080", 12, 0, 4);
  g.fillRect(w / 2 - 30, h / 2 - 30, 60, 60, "ffffff", 8);
  g.restore();
  g.text("Hello", 12, h - 14, { size: 22, color: "ffffff" });
}, onKeyDown: (e) => {
  if (e.key === "Left" || e.key === "Right") { angle += e.key === "Left" ? -5 : 5; return true; }
} })
```

- **Pictures:** `ui.image(src, {height, fit: "contain" | "cover" | "fill", radius, opacity})` and `g.image(src, x, y, w, h)` on a canvas. `src` is a file of the extension (`"images/logo.png"` , from the package or the folder) or an absolute path the extension may read. PNG, JPEG, BMP, GIF, TGA, PSD and SVG; each picture is decoded once. `ui.imageSize(src)` gives its size in pixels.
- **Colours** are theme names (`"accent"` , `"text"` , `"text2"` , `"text3"` , `"raised"` , `"hairline"` , ...), `"#rrggbb"` , or `[r, g, b, a]` ; `hope.ui.theme` returns the current ones.

10. Undo, events and storage

- **Undo:** every callback is one undo step named after the extension. A drag on a slider, a number or the canvas is one step for the whole drag.
- **Events:** `hope.on(name, fn)` / `hope.off(name, fn)` . They also arrive while the panel isn't shown, and what a handler changes is one undo step.

Event	When (arguments)
<code>selectionChanged</code>	other layers selected (their ids)
<code>keySelectionChanged</code>	other keyframes selected in the Timeline (read <code>comp.selectedKeys</code>)
<code>timeChanged</code>	the playhead moved (seconds)
<code>compChanged</code>	another comp is open (its id)
<code>projectChanged</code>	any edit, an undo, another project
<code>projectOpened</code> / <code>projectSaved</code>	another project; the project saved (its path)
<code>playbackStarted</code> / <code>playbackStopped</code>	playback (the playhead's time)
<code>exportStarted</code> / <code>exportFinished</code>	an export (<code>{comp, file}</code>); at the end <code>{ok, file, error}</code>)
<code>themeChanged</code>	Dark / Light switched (<code>{light}</code>)

- **Storage:** `hope.storage.get(key, default)` / `set(key, value)` keep values between sessions, per extension.

- **Files:** `hope.ui.chooseFile()` lets the user pick a file the extension may then read or write. To read and write files freely an extension asks in its manifest: `"permissions": ["files"]`. Files that ship in your package are read with `hope.extension.read("presets/default.json")`.

11. Timers, commands and dialogs

- **Timers:** `setTimeout`, `setInterval`, `clearTimeout`, `clearInterval` (also as `hope.setTimeout ...`). They run while the panel is hidden too. Each call is one undo step, and the panel renders again after it. A timer that throws stops, and its error shows in the panel. Menu scripts have no timers: they run once, from start to end.
- **Commands:** `hope.addCommand(id, name, run, {shortcut})` puts a command into **Tools ▸ Extension Commands ▸ <your extension>** and into **Edit ▸ Keyboard Shortcuts**, where users can give it any key. A shortcut you ask for (`"Mod+Shift+K"`, where Mod is Ctrl on Windows and ⌘ on macOS) is only given if no other action has it. The answer `{shortcut, takenBy}` says what it got. Running a command is one undo step named after it. `hope.removeCommand(id)` takes it out again.
- **Dialogs:** `hope.ui.confirm({title, message, ok}, (yes) => ...)` and `hope.ui.prompt({title, value, placeholder, ok}, (text) => ...) (null on Cancel)`. The answer comes as a callback, so the app never waits on a script.

```
hope.addCommand("flip", "Flip Selected Layers", () => {
  for (const l of hope.activeComp.selectedLayers) l.property("Scale").setValue([-100, 100]);
}, { shortcut: "Mod+Alt+F" });

let saved = 0;
setInterval(() => { saved++; hope.ui.refresh(); }, 1000); // a clock in the panel

hope.on("exportFinished", (r) => { if (r.ok) hope.alert("Rendered " + r.file); });
hope.ui.prompt({ title: "Rename", value: "Title" }, (name) => {
  if (name !== null) hope.activeComp.selectedLayers.forEach((l) => l.name = name);
});
```

12. Developer Mode

Developer ▸ Developer Mode loads extensions in development and reloads a panel as soon as its script file changes. The panel's right-click menu then offers **Reload Extension** and **Show Extension Folder**, and errors appear in full, with their line, at the bottom of the panel. Everything else for developers is in the **Developer** menu too.

13. Packaging and signing (.hopex)

Extensions ship as one `.hopex` file, signed with your certificate. Inside the package the code is minified and encrypted, and Hope Creative never unpacks it to disk. `hopex-sign` is in the Plug-in SDK folder (**Developer ▸ Developer Resources ▸ Open SDK and Examples**).

```
hopex-sign cert create --name "Your Name" --organization "Your Studio" --email you@example.com --
password <pw> --out me.hopecert
hopex-sign pack MyExtension --out MyExtension.hopex --cert me.hopecert --password <pw>
hopex-sign verify MyExtension.hopex
```

- **Keep the certificate and its password safe.** Updates should be signed with the same one.

- **Installing:** users drop the `.hopex` on Hope Creative or use **Developer ▸ Extensions ▸ Install Extension...** The install dialog shows who signed it and what it may do. Installed packages are under **Developer ▸ Extension Folders ▸ Installed; Developer ▸ Extensions ▸ Uninstall** removes one.
- **What gets refused:** a package changed after signing is never installed. An unsigned one installs only in Developer Mode.

14. Licenses for paid extensions

Your license server (or you, by hand) signs license data with your certificate. The extension checks it with your public key (`hopex-sign key me.hopercert`):

```
hopex-sign license --cert me.hopercert --password <pw> --data '{"user": "Ada", "machine": "<their machine id>"}' --out license.json
```

```
const PUBLIC_KEY = "ORxC..."; // hopex-sign key me.hopercert
const result = hope.license.verify(hope.storage.get("license", ""), PUBLIC_KEY);
if (!result.valid) { /* result.reason: not signed by you, for another computer ... */ }
// hope.license.machineId: this computer's id (to bind a license to it)
```

15. Examples

Developer ▸ Developer Resources ▸ Open SDK and Examples opens the folder with:

- **Script Examples:** a promo title, staggering, renaming and shaking the selected layers, a lower third. Copy one into the Scripts folder to see it in **Tools ▸ Scripts**.
- **Extension Examples:**
 - `curve-animator` rebuilds the built-in Curve Animator: `ui.curve`, preset chips, a name field, storage, and keyframe editing through `comp.selectedKeys`;
 - `element-gallery` shows every element;
 - `layer-tools` renames, centres and staggers the selected layers.

Copy one into `Documents\Hope Creative\Extensions` and turn on Developer Mode.