

Extending Storyline with JavaScript

One-page quick reference for Articulate Storyline 360 developers. Drawn from published work: The High-Risk Offboarding and A Healthcare Patient Journey.

1. When to reach for JavaScript

- ✓ **Cross-variable logic** in one action, such as reading four variables to build one personalized output
- ✓ **DOM-level control** of elements the trigger panel can't reach (hover states, dynamic text, animation timing)
- ✓ **Third-party libraries** such as GSAP for animation beyond built-in motion paths
- ✓ **Dynamic, generated content** that changes with a combination of inputs

Rule of thumb: about to build 8+ duplicate slides to cover permutations of 3 to 4 variables? Write one JavaScript block instead.

2. Targeting elements

Storyline gives each object a `data-model-id` in the DOM. Target with

`document.querySelector('[data-model-id="..."]')`, not the object's display name.

```
const player = GetPlayer();
const risk = player.GetVar("riskScore");
const tone = player.GetVar("conversationTone");
// compose output from all variables
```

`GetPlayer()` and `GetVar()` are the bridge from Storyline's variables to custom code.

3. WATCH OUT: data-model-id can regenerate on republish

JavaScript that hard-codes an ID **breaks silently** after republishing. No error appears, the element just no longer exists.

- Avoid raw IDs in production without a fallback; prefer a stable property or a custom attribute you control.
- If you must use an ID, re-verify it after **every** republish. Treat it as a required QA step.
- Document which slides and objects depend on JS-targeted IDs so the QA pass knows what to recheck.

4. Common patterns

Hover shadow / dynamic style

Attach an event listener to a targeted element to toggle a CSS class or inline style on hover or click.

Variable bridge

Read variables with `GetVar()`, apply logic the trigger panel can't (multi-variable conditions, string composition), optionally write back with `SetVar()`.

GSAP flip / advanced animation

Add GSAP as a Web Object or injected script, then fire it from a Storyline JS trigger tied to a slide event.

Web Object wrapper

Build a custom element (game, visualization) as standalone HTML/CSS/JS, embed it, and bridge state back into variables so scores and completion still report.

5. Debugging workflow

1. **Publish to Review 360 or a local web export**, not just Preview. Editor preview doesn't always match published output.
2. **Open browser dev tools** (Console and Elements) and confirm the `data-model-id` you target exists and matches your script.
3. **Log at each stage** of variable reads and writes with `console.log`. Remove or comment out before final publish.
4. **Republish once during QA** and re-verify every JS-dependent element (see section 3).
5. **Isolate the side:** if a variable isn't updating, check whether the JS runs at all (a misfiring trigger) before blaming the logic.