```
// Pavilion browser presentation example
// Replace each bracketed placeholder before using this in your World.
// [WORLD_SLUG]: a unique short identifier for this World's state and messages.
// [SCREEN_ANCHOR_NAME]: the Unity GameObject that anchors the panel.
// [BROWSER_OBJECT_NAME]: the GameObject with the BS Browser component.
// [OPTIONAL_PLACEHOLDER_NAME]: an optional image shown when no URL is set.
// Adjust panel localPosition and localScale for your own World layout.
// The Host account must pass the SDK's isSpaceAdmin permission check.

/* World presentation example for SideQuest Creator SDK 4.x. */
(() => {
    "use strict";
    if (window.presentationInstalled) return;
    window.presentationInstalled = true;
    const KEY = "[WORLD_SLUG].presentation.v1";
    const clamp = n => Math.max(0, Math.min(1, Number(n) || 0));

    function webUrl(value) {
        const text = String(value || "").trim();
        if (!text) throw new Error("Paste a document or website URL first.");
        const url = new URL(/^[a-z][a-z0-9+.-]*:/i.test(text) ? text : "https://" +
        text);
        if (!/^https?:$/.test(url.protocol) || url.username || url.password)
            throw new Error("Use an http or https URL without embedded login details.");
        return url.href;
    }

    // Runs inside the destination page, not an iframe wrapper. Google document
    // viewers often scroll a nested viewport instead of document.scrollingElement.
    function scrollPage(fraction) {
        let target = document.scrollingElement || document.documentElement;
        let area = 0;
        for (const element of document.querySelectorAll("*")) {
            if (element.clientHeight < 120 || element.scrollHeight <=
            element.clientHeight + 8) continue;
            const style = getComputedStyle(element);
            if (!/auto|scroll/.test(style.overflowY)) continue;
            const rect = element.getBoundingClientRect();
            const visible = Math.max(0, Math.min(innerWidth, rect.right) - Math.max(0,
            rect.left)) *
                Math.max(0, Math.min(innerHeight, rect.bottom) - Math.max(0, rect.top));
            if (visible > area) { area = visible; target = element; }
        }
        target.scrollTop = Math.max(0, target.scrollHeight - target.clientHeight) *
        fraction;
    }

    // Runs in the presenter's browser page, including after same-window navigation.
    function observePresenterPage() {
        if (window.__presentationObserved) return;
        window.__presentationObserved = true;
        const send = data => {
            if (window.O?.sendMessage) window.O.sendMessage("app", JSON.stringify({
            source: "[WORLD_SLUG]-presentation", ...data }));
        };
        send({ kind: "url", url: location.href });
        document.addEventListener("click", event => {
            send({ kind: "click", x: event.clientX, y: event.clientY, url:
            location.href });
        }, true);
        let lastScroll = 0;
        document.addEventListener("scroll", event => {
            if (Date.now() - lastScroll < 180) return;
            lastScroll = Date.now();
            const target = event.target === document ? document.scrollingElement :
            event.target;
```

```
            if (!target || target.scrollHeight <= target.clientHeight) return;
            send({ kind: "scroll", fraction: target.scrollTop / (target.scrollHeight -
            target.clientHeight) });
        }, true);
        window.addEventListener("popstate", () => send({ kind: "url", url:
        location.href }));
        window.addEventListener("hashchange", () => send({ kind: "url", url:
        location.href }));
    }

    function start() {
        if (window.presentationStarted) return;
        window.presentationStarted = true;
        const scene = BS.Scene.GetInstance();
        let started = false;
        async function ready() {
            if (started) return;
            const anchor = await scene.Find("[SCREEN_ANCHOR_NAME]");
            if (!anchor) { console.error("The World screen anchor is missing. Add the
            screen anchor in Unity."); return; }
            started = true;
            const placeholder = await scene.Find("[OPTIONAL_PLACEHOLDER_NAME]");
            if (placeholder) placeholder.SetActive(true);
            const browserObject = await scene.Find("[BROWSER_OBJECT_NAME]");
            const browser = browserObject?.GetComponent(BS.CT.Browser);
            if (!browser) { console.error("The presentation's browser object is missing
            its BS Browser component."); return; }
            let state = { url: browser.url || "", scroll: 0, blank: false, revision: ""
            };
            let pendingState = null;
            let generation = 0;
            const host = () => scene.localUser?.isSpaceAdmin === true;
            const panelObject = new BS.GameObject({ name: "Presentation Controls",
            parent: anchor,
                localPosition: new BS.Vector3(-3.05, -1.45, -0.5),
                localScale: new BS.Vector3(0.18, 0.18, 0.18) });
            const panel = new BS.UI(new BS.Vector2(1000, 420));
            await panelObject.AddComponent(panel);
            const controls = [];
            let field, status;
            let draftUrl = state.url;

            function action(script) {
                if (browser) browser.RunActions(JSON.stringify({ actions: [{
                actionType: "runscript", strParam1: script }] }));
            }
            function applyScroll() {
                action("(" + scrollPage.toString() + ")(" + clamp(state.scroll) + ");");
            }
            function statusText(text) { if (status) status.text = text; }
            function updateUI() {
                controls.forEach(control => { control.enabled = host(); });
                if (field) field.isReadOnly = !host();
                browser.ToggleInteraction(host() && !state.blank);
                statusText(host() ? (!state.url ? "Banner displayed. Paste a URL and
                press Open." : state.blank ? "Screen blank. Press Blank / Show to
                resume." : "Presenting: " + state.url) : "Following the presenter. The
                Host controls this screen.");
            }
            async function apply(next) {
                if (!next || typeof next.url !== "string" || typeof next.blank !==
                "boolean") return;
                if (next.url) { try { next.url = webUrl(next.url); } catch (_) {
                return; } }
                const urlChanged = next.url !== state.url;
                const navigate = urlChanged || next.revision !== state.revision;
```

```
113                 const changedScroll = clamp(next.scroll) !== state.scroll;
114                 state = { ...next, scroll: clamp(next.scroll) };
115                 if (placeholder) placeholder.SetActive(!state.url);
116                 if (field && navigate) { field.value = state.url; draftUrl = state.url; }
117                 if (state.blank || !state.url) {
118                     ++generation;
119                     browserObject.SetActive(false);
120                     updateUI();
121                     return;
122                 }
123                 browserObject.SetActive(true);
124                 if (urlChanged) {
125                     browser.url = state.url;
126                 } else if (navigate) {
127                     action("location.reload();");
128                 }
129                 if (navigate || changedScroll) {
130                     const current = ++generation;
131                     // Component readiness is not page readiness. Retry briefly for
                        slow viewers;
132                     // the generation guard prevents an old page's timers moving a new
                        page.
133                     [150, 700, 1800, 4000, 8000].forEach(delay => setTimeout(() => {
134                         if (current === generation && browser) {
                            browser.ToggleInteraction(host()); applyScroll(); }
135                     }, delay));
136                 }
137                 updateUI();
138             }
139             function publish(patch) {
140                 if (!host()) { statusText("Only the Host can present."); return; }
141                 // A local change is not proof that the space accepted the write.
142                 // Build rapid edits on the latest requested state, then display only
143                 // the state confirmed by space-state-changed on every client.
144                 pendingState = { ...(pendingState || state), ...patch };
145                 statusText("Syncing presentation...");
146                 scene.SetProtectedSpaceProps({ [KEY]: JSON.stringify(pendingState) });
147             }
148             browserObject.On("browser-message", event => {
149                 if (!host()) return;
150                 let message;
151                 try { message = JSON.parse(event.detail); } catch (_) { return; }
152                 if (message?.source !== "[WORLD_SLUG]-presentation") return;
153                 if (message.kind === "url" && typeof message.url === "string" &&
                    message.url !== state.url) {
154                     try { publish({ url: webUrl(message.url), scroll: 0, revision:
                        String(Date.now()) }); } catch (_) { /* Ignore internal browser
                        pages. */ }
155                 } else if (message.kind === "scroll" &&
                    Number.isFinite(message.fraction) && Math.abs(clamp(message.fraction) -
                    state.scroll) > 0.01) {
156                     publish({ scroll: clamp(message.fraction) });
157                 } else if (message.kind === "click" && Number.isFinite(message.x) &&
                    Number.isFinite(message.y)) {
158                     scene.OneShot({ type: "[WORLD_SLUG]-presentation-click", x:
                        message.x, y: message.y, url: message.url });
159                 }
160             });
161             scene.On("one-shot", event => {
162                 let message;
163                 try { message = JSON.parse(event.detail.data); } catch (_) { return; }
164                 if (host() || message?.type !== "[WORLD_SLUG]-presentation-click" ||
                    state.blank || message.url !== state.url) return;
165                 const sender = Object.values(scene.users).find(user => user.id ===
                    event.detail.fromId);
166                 if (!sender?.isSpaceAdmin || !Number.isFinite(message.x) ||
```

```
            !Number.isFinite(message.y)) return;
        const x = Math.max(0, Math.min(browser.pageWidth, message.x));
        const y = Math.max(0, Math.min(browser.pageHeight, message.y));
        action("document.elementFromPoint(" + x + "," + y + ")?.click();");
    });
    // Reinstall on each navigated page. Only the presenter sends events.
    setInterval(() => {
        if (host() && !state.blank) action("(" +
        observePresenterPage.toString() + ")();");
    }, 1200);
    function buildControls() {
        panel.SetBackgroundColor(new BS.Vector4(0.04, 0.085, 0.075, 1));
        const title = panel.CreateLabel("PRESENTATION");
        title.SetStyles({ fontSize: "30px", color: "#ffffff", marginBottom:
        "12px" });
        const urlLabel = panel.CreateLabel("Website address");
        urlLabel.SetStyles({ fontSize: "20px", color: "#d9eee4", marginBottom:
        "4px" });
        field = new BS.UITextField(panel);
        field.placeholder = "Type a website address";
        field.maxLength = 4096;
        field.SetStyles({ height: "64px", fontSize: "24px", color: "#10241c",
            backgroundColor: "#f0f7f2", marginBottom: "12px" });
        field.OnChange(event => { draftUrl = String(event.value ?? field.value
        ?? ""); });
        status = panel.CreateLabel("");
        status.SetStyles({ fontSize: "20px", color: "#ffffff", whiteSpace:
        "normal", marginBottom: "8px" });
        const row = panel.CreateVisualElement();
        row.SetStyles({ flexDirection: "row", flexWrap: "wrap" });
        function button(text, handler) {
            const control = panel.CreateButtonWithText(text, text, row);
            control.SetStyles({ height: "54px", minWidth: "135px", fontSize:
            "22px",
                marginRight: "8px", marginBottom: "8px", color: "#ffffff",
                backgroundColor: "#24584b" });
            control.OnClick(() => { try { handler(); } catch (error) {
            statusText(error.message); } });
            controls.push(control);
        }
        button("Clear URL", () => { field.value = ""; draftUrl = "";
        field.Focus(); });
        button("Open", () => publish({ url: webUrl(draftUrl), scroll: 0, blank:
        false, revision: String(Date.now()) }));
        button("Reload", () => { if (state.url) publish({ revision:
        String(Date.now()) }); });
        button("Scroll up", () => publish({ scroll: clamp(state.scroll - 0.06)
        }));
        button("Scroll down", () => publish({ scroll: clamp(state.scroll +
        0.06) }));
        button("Top", () => publish({ scroll: 0 }));
        button("Blank / Show", () => { if (state.url) publish({ blank:
        !state.blank }); });
        const help = panel.CreateLabel("Use a view link attendees can access.
        Pages load separately for each attendee.");
        help.SetStyles({ fontSize: "18px", color: "#c4d8cf", whiteSpace:
        "normal", marginTop: "8px" });
        field.value = state.url;
        draftUrl = state.url;
        updateUI();
    }
    buildControls();
    scene.On("space-state-changed", event => {
        for (const change of event.detail.changes || []) {
            if (change.property !== KEY || change.isPublic === true ||
            change.type === "public") continue;
```

```
                    try {
                        const next = JSON.parse(change.newValue);
                        if (pendingState && JSON.stringify(next) ===
                        JSON.stringify(pendingState)) pendingState = null;
                        apply(next);
                    } catch (_) { statusText("Could not read the presentation state."); }
                }
            });
            scene.On("space-state-error", event => {
                if (event.detail?.key !== KEY) return;
                pendingState = null;
                statusText(event.detail.code === "not_authorized"
                    ? "Could not sync. Use a Host account with World owner or moderator
                    permission."
                    : "Could not sync the presentation change. Please try again.");
                console.warn("Presentation state was rejected:", event.detail.code,
                event.detail.message);
            });
            scene.On("user-joined", updateUI);
            const saved = scene.spaceState?.protected?.[KEY];
            if (saved) { try { apply(JSON.parse(saved)); } catch (_) { updateUI(); } }
            else { field && (field.value = state.url); updateUI(); }
        }
        if (scene.unityLoaded) ready(); else scene.On("unity-loaded", ready);
    }
    window.addEventListener("bs-loaded", start, { once: true });
    if (window.BS?.Scene) start();
})();

```