

devs3 Developer Platform 2.0

Game Developer Handbook

Version 1.0 · A practical guide for studios, solo developers, publishers, and technical teams preparing an HTML5 game for a web-game platform.

How to use this handbook

This handbook covers the complete game-delivery workflow: preparing a build, creating a game listing, testing, submitting, releasing, operating, and improving a game. It is designed to be useful before an account has access to every optional platform service.

Some values are issued per game in the Developer Portal, such as the game ID, upload destination, SDK configuration, ad configuration, and release status. Copy those values from your portal rather than reusing examples from another project. Never place private keys, server secrets, or unreviewed third-party credentials in a browser game build.

1. Before you begin

1.1 Decide the release owner

Use one organisation owner for every submitted game. The owner should control the developer account, billing or tax information when applicable, source repository access, store assets, and the signing or deployment process. Add collaborators using the least access they need.

1.2 Prepare these materials

- A production build that starts from an `index.html` entry point.
- A unique game title, short description, long description, genre, and supported languages.
- A square icon, landscape and portrait screenshots where relevant, and a cover or promotional image.
- A contact email for release and support questions.
- A privacy-policy URL if the game or its vendors collect personal data, use cookies, advertising, analytics, cloud saves, or account sign-in.
- A tested fallback experience for slow connections, ad blockers, paused tabs, and lost focus.

1.3 Choose a technical baseline

HTML5 games should work in current desktop and mobile browsers. Test mouse, touch, keyboard, responsive resize, fullscreen changes, audio unlock on first user interaction, and orientation changes. Keep all resource URLs relative to the uploaded game package unless a platform-provided URL is explicitly required.

2. Developer Portal workflow

2.1 Create your developer profile

Complete the organisation or developer profile before submitting a game. Use a real business or developer name, a monitored email address, support contact details, and team member roles. Keep release contacts current so a reviewer can resolve a blocking issue quickly.

2.2 Create a game project

Create one project per game. Use a stable internal name and a public display name. The portal normally keeps draft, submitted, approved, published, suspended, and archived states separate. A draft is not visible to players.

Record these project values in your team's private release notes:

Value	Why it matters
Game ID	Identifies the game in portal settings, reporting, SDK configuration, and support requests.
Release channel	Separates test or staging builds from the public release.
Current version	Lets the team trace defects and roll back safely.
Support contact	Lets the platform reach the correct person for review or incident questions.

2.3 Add store metadata

Write player-facing metadata before submission. The title, description, tags, category, age suitability, controls, and screenshots must accurately represent the build. Do not use unrelated brands, misleading screenshots, keyword stuffing, or promises that the game does not deliver.

Good short description example: "A quick puzzle game where players connect colour paths before the timer ends."

3. Build and package your HTML5 game

3.1 Required package shape

Start with a clean, self-contained release folder. The exact upload method is shown in the Developer Portal, but the build itself should be structured like this:

```
my-game-release/  
index.html  
assets/  
js/  
css/  
manifest.json (optional)  
README-release.txt (recommended)
```

The entry file must load without local development tools, a development server, a source-map server, or credentials stored in the browser. Remove test pages, unused packages, private `.env` files, source-control folders, and server-side secrets before packaging.

3.2 Build checklist

1. Produce a clean production build.
2. Open `index.html` through the same type of hosted environment used for release.
3. Verify all JavaScript, CSS, fonts, images, audio, and data files load successfully.
4. Check browser console for errors, mixed-content warnings, and failed network requests.
5. Test a cold start, restart, pause/resume, tab switch, and a slow-network session.

6. Make a compressed upload archive only if the portal specifically requests one.

3.3 Performance targets

Use compressed images and audio, lazy-load optional assets, keep the initial download small, and avoid blocking the main thread during startup. Measure first interaction, memory use, frame rate, and recovery after a pause. Test on a lower-powered mobile device, not only a desktop development machine.

4. Platform integration

4.1 Use project-specific integration settings

If the portal enables an SDK, copy its project-specific configuration from the project's Integration or SDK page. Do not hard-code a configuration copied from another game. Keep server-only secrets on your server; browser builds may use only values documented as public.

4.2 Lifecycle events

Games with a platform SDK commonly need to cooperate with page and ad lifecycle. Your game should have explicit functions to pause and resume safely:

```
function pauseGame() {  
  gameLoop.stop();  
  audio.pause();  
}  
  
function resumeGame() {  
  audio.resumeAfterUserGesture();  
  gameLoop.start();  
}
```

Connect these functions to the current project's documented SDK callbacks. Treat callbacks as optional until the SDK is available, and ensure the game remains playable if a non-essential service is unavailable.

4.3 Scores, achievements, and leaderboards

Only submit a score after the game has validated the run. Use the game ID and score format configured for that project. Never trust an arbitrary score value directly from a client when rewards, contests, or rankings depend on it. Store enough run context to investigate suspicious activity.

Before enabling a leaderboard, define:

- Whether higher or lower values are better.
- The score range and integer/decimal format.
- When a score is final.
- How ties are resolved.
- Whether a season, tournament, or daily reset applies.

4.4 Analytics events

Use a small, stable event taxonomy. Typical events are `game_loaded`, `game_started`, `level_started`, `level_completed`, `game_over`, `tutorial_completed`, `ad_requested`, and `ad_completed`. Send only data allowed by your privacy notice and the platform configuration. Do not attach raw email addresses, phone numbers, or other direct identifiers to analytics events.

5. Advertising and monetisation readiness

Advertising, payments, or revenue sharing are project-specific and may require review before activation. Configure only units and formats assigned in the Developer Portal.

5.1 Ad-safe game behaviour

- Pause gameplay before an interstitial or rewarded-ad break and restore it after the documented completion callback.
- Do not promise an in-game reward until the rewarded-ad completion signal is received.
- Keep a non-ad fallback path when an ad is unavailable, blocked, or times out.
- Do not trigger ads from accidental taps, loading screens, or a loop that repeats requests.
- Respect consent and privacy choices before requesting personalised advertising where required.

5.2 Revenue reporting

Use portal reports as the source of truth for platform revenue, adjustments, and payment status. Reconcile game version, date range, locale, and reporting time zone before comparing your own analytics to platform figures. Never expose financial data in the public game client.

6. Quality assurance

6.1 Functional test matrix

Test the release build on current Chrome, Edge, Firefox, and Safari where supported, plus Android Chrome and iOS Safari when the game is mobile compatible. Test at least one narrow viewport and one wide desktop viewport.

Scenario	Expected result
Fresh load	A clear loading state and a playable game without console errors.
First interaction	Audio and controls activate only after a user gesture when needed.
Resize/orientation	Canvas and controls remain visible and usable.
Background/return	Game pauses safely and resumes without duplicate loops or audio.
Network delay	Player sees a usable loading or retry state, not a frozen blank screen.
Game over/restart	Score resets correctly and no duplicate analytics or ad requests occur.
Ad unavailable	Gameplay continues or a documented fallback is shown.

6.2 Accessibility and player safety

Provide understandable controls, sufficient contrast, visible focus states for keyboard navigation where applicable, readable text at common viewport sizes, and a way to mute or reduce audio. Avoid content that is deceptive, hateful, sexually exploitative, infringing, malicious, or unsafe for the stated audience.

7. Submit for review

7.1 Submission checklist

1. Upload the production build using the project's upload area.
2. Confirm the launch URL or uploaded entry point opens in the portal preview.
3. Complete title, description, categories, tags, icon, screenshots, and age information.
4. Add support and privacy-policy information when required.
5. Test all enabled integrations in the preview or test channel.
6. Add concise reviewer notes: test credentials if any, control instructions, special device requirements, and known limitations.
7. Submit the exact version you tested.

7.2 Common review blockers

- Broken entry file, missing assets, or a build that only works on localhost.
- A blank screen, fatal console error, infinite loading state, or unresponsive controls.
- Misleading listing assets or incomplete game information.
- External resources that fail over HTTPS or require unauthorised access.
- Privacy disclosures missing for enabled data collection or advertising.
- Ads or reward flows that interrupt the game incorrectly.
- Unlicensed assets, copyrighted material, or content-policy concerns.

8. Release management

8.1 Publish safely

Publish only after the approved build matches the tested version. Keep the prior production archive and a release note containing version, date, changes, asset hash if available, owner, and rollback steps.

8.2 Update workflow

Use this repeatable release flow:

1. Create a new build version and change log.
2. Test it in the project's test or preview channel.
3. Validate gameplay, lifecycle, analytics, and enabled monetisation.
4. Submit the update for any required review.
5. Publish during a monitored window.
6. Watch error reports, load failures, player feedback, and key engagement metrics.
7. Roll back or disable a feature if a critical issue appears.

8.3 Incident response

If the public game fails, first stop further impact: disable the affected release or revert to the last known-good version when the portal permits. Capture the game ID, version, browser/device, timestamp, screenshot, console error, and reproduction steps before contacting support.

9. Operating your game after launch

9.1 Monitor meaningful metrics

Track starts, completion or game-over rate, session duration, repeat play, crash or error rate, load failures, and retention. Interpret metrics alongside releases, traffic sources, device mix, consent rates, and ad availability. A single metric is not a complete measure of game quality.

9.2 Manage player feedback

Use a monitored support channel. Categorise reports as gameplay bug, performance, account or score issue, payments/ads, safety, or feature request. Ask for device, browser, game version, and a reproduction path; never ask players to send passwords or authentication tokens.

9.3 Keep assets and dependencies healthy

Review third-party libraries, audio/image licences, external CDNs, privacy links, and support contact details regularly. Remove no-longer-needed analytics and advertising integrations. Test the game after browser-engine updates or major dependency upgrades.

10. Security and privacy checklist

- Keep secrets, signing keys, private API tokens, and privileged endpoints off the client.
- Validate all player-controlled input before storing or displaying it.
- Use HTTPS for every external resource.
- Restrict third-party scripts to services your team has reviewed.
- Publish an accurate privacy policy before collecting data or enabling relevant third-party services.
- Honour consent choices and data-deletion requests according to your applicable obligations and platform settings.
- Keep an internal record of what data the game collects, why, where it is sent, and how long it is retained.

11. Support request template

Use this format when contacting platform support:

Game title:
Game ID:
Release channel and version:
Issue summary:
When it started (date/time/time zone):
Affected browser/device:
Steps to reproduce:
Expected result:
Actual result:
Console or network error:
Screenshot/video:
Business impact:

12. Final pre-launch checklist

- ■ Project ownership and support contacts are correct.

- ■ Release build opens from its uploaded entry point.
- ■ All assets load over HTTPS with no fatal console errors.
- ■ Desktop, mobile, resize, orientation, pause, and restart flows were tested.
- ■ Metadata, icon, screenshots, age information, and descriptions are accurate.
- ■ Score, analytics, SDK, and advertising integrations were tested only where enabled.
- ■ Privacy, consent, and third-party disclosures are complete for the released configuration.
- ■ Prior working build and rollback instructions are retained.
- ■ Team will monitor the release after publishing.

Where to find the current platform-specific values

The Developer Portal is the source of truth for your game's upload method, project ID, SDK setup, enabled services, review status, and policy notices. Use the in-portal documentation and support channel for any feature that is not enabled for your project. This handbook intentionally does not include reusable credentials, private endpoints, or a copied integration key.