# Game Integration Guide

Economy cohesion + metrics + run grants (ArcadeBridge spec)

**Version:** v1.1 **Last updated:** 2026-01-05

This document is generic by design: you can apply it to any game genre, including economy-heavy games that spend Credits during gameplay.

**Golden rule:** In Arcade embeds, the parent owns the economy and the leaderboard. The game is a client.

# 1. Key concepts

**Credits (AC)**: Off-chain unit anchored to $0.01 for UX. Users buy Credits on-chain; games do not own balances.

**Run grants**: Embedded games must gate gameplay start behind a grant from the Arcade parent. No grant = no eligible run.

**Multi-metric leaderboards**: Each game can expose multiple metrics. The catalog defines what metrics exist and how to rank them.

**Fairness guardrails** (economy-heavy games): track inRunSpendAC, enforce rankedSpendCapAC, optionally split ranked vs sandbox.

# 2. Minimum viable integration flow

| Mode | What happens |
|---|---|
| Embedded (iframe) | READY → REQUEST_RUN on first start input → wait RUN_GRANTED → play → RUN_RESULT o |
| Standalone (no parent) | Start immediately (demo mode). Still track duration + metrics locally. |

If the first start input is also an action (jump/flap/thrust), queue that input and apply it immediately after RUN_GRANTED for “instant feel”.

# 3. ArcadeBridge message protocol (v1)

All messages are sent via postMessage and must include a shared channel string and the gameId.

## 3.1 Outbound messages (game → parent)

**ARCADE:READY** (send once on load)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:READY",
  "gameId": "moonshot",
  "payload": { "version": "1.0.0" }
}
```

**ARCADE:REQUEST_RUN** (send on first start intent; do not start gameplay yet)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:REQUEST_RUN",
  "gameId": "moonshot",
  "payload": {
    "mode": "ranked",
    "requestedCostAC": 0,
    "wantsEligible": true
  }
}
```

**ARCADE:RUN_RESULT** (send once at game over)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:RUN_RESULT",
```

```
  "gameId": "stormhouse2",
  "payload": {
    "runId": "run_01HS...",
    "durationMs": 183420,
    "metrics": {
      "waves": 23,
      "kills": 812,
      "accuracyBp": 7345,
      "efficiency": 1.72,
      "inRunSpendAC": 42,
      "score": 912345
    }
  }
}
```

Optional vNext (recommended for economy-heavy games): **ARCADE:SPEND_REQUEST** so the parent approves each in-run spend event.

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:SPEND_REQUEST",
  "gameId": "stormhouse2",
  "payload": {
    "runId": "run_01HS...",
    "amountAC": 5,
    "sku": "upgrade:rifle_1",
    "meta": { "level": 1 }
  }
}
```

## 3.2 Inbound messages (parent → game)

**ARCADE:SYNC** (balances / identity snapshot; render-only)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:SYNC",
  "gameId": "moonshot",
  "payload": {
    "walletConnected": true,
    "pohVerified": false,
    "paidAC": 1200,
    "promoAC": 50,
    "pro": { "active": true, "tier": 2, "expiresAt": 1760000000 },
    "identity": { "displayName": "NovaPilot", "avatarPng": "https://.../avatar.png" }
  }
}
```

**ARCADE:RUN_GRANTED** (authorizes gameplay start)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:RUN_GRANTED",
  "gameId": "moonshot",
  "payload": {
    "runId": "run_01HS...",
    "runType": "paid",
    "eligible": true,
    "costAC": 5,
    "rankedSpendCapAC": 25,
    "metric": "score"
```

```
  }
}
```

**ARCADE:RUN_DENIED** (show reason + hint)

```
{
  "channel": "ARCADE_BRIDGE_V1",
  "type": "ARCADE:RUN_DENIED",
  "gameId": "moonshot",
  "payload": {
    "reason": "not_connected|insufficient_funds|poh_required|rate_limited",
    "message": "Connect wallet to play."
  }
}
```

# 4. Metrics design (generic)

Metrics let your game be scored fairly across genres. Define metrics in the catalog so the Arcade UI and backend can treat them consistently.

## Metric fields

Each metric has: id, label, type (int/float), direction (desc/asc), plus optional clamp and rounding rules.

## Recommended metric patterns

- Score (desc): arcade classics, shooters, endless runners
- Time (asc): racing, speedruns, puzzles
- Waves/Level reached (desc): defense, roguelikes
- AccuracyBp (desc): shooters (basis points 0..10000)
- Efficiency (desc): output / spend (kills per AC, DPS per AC)
- inRunSpendAC (desc, capped for ranked): how much was spent during the run

Payout eligibility: Skill payouts use the **paid-only eligible** leaderboards. “Fun boards” can include free/promo runs, but payouts do not.

# 5. Embedded UX requirements

- On start input, show “Requesting run…” immediately.
- If no response in 2.5s, show a clear fallback message (connect wallet / not enough credits).
- If RUN_DENIED arrives, display its message and a hint (“Open Wallet”).
- Standalone mode must remain playable as a demo.

# 6. Security and validation

Prefer a specific parent origin derived from document.referrer. Always require channel + gameId match before acting.

## QA checklist

- Standalone mode works without parent.
- Embedded mode sends READY and requests a run before starting.
- RUN_GRANTED starts exactly one run and assigns a runId.
- RUN_RESULT sends exactly once and includes durationMs + metrics.
- Metrics are numeric and within expected ranges.
- If the game spends Credits in-run, enforce rankedSpendCapAC when eligible.