

ERAGANX / DEVELOPER GUIDE

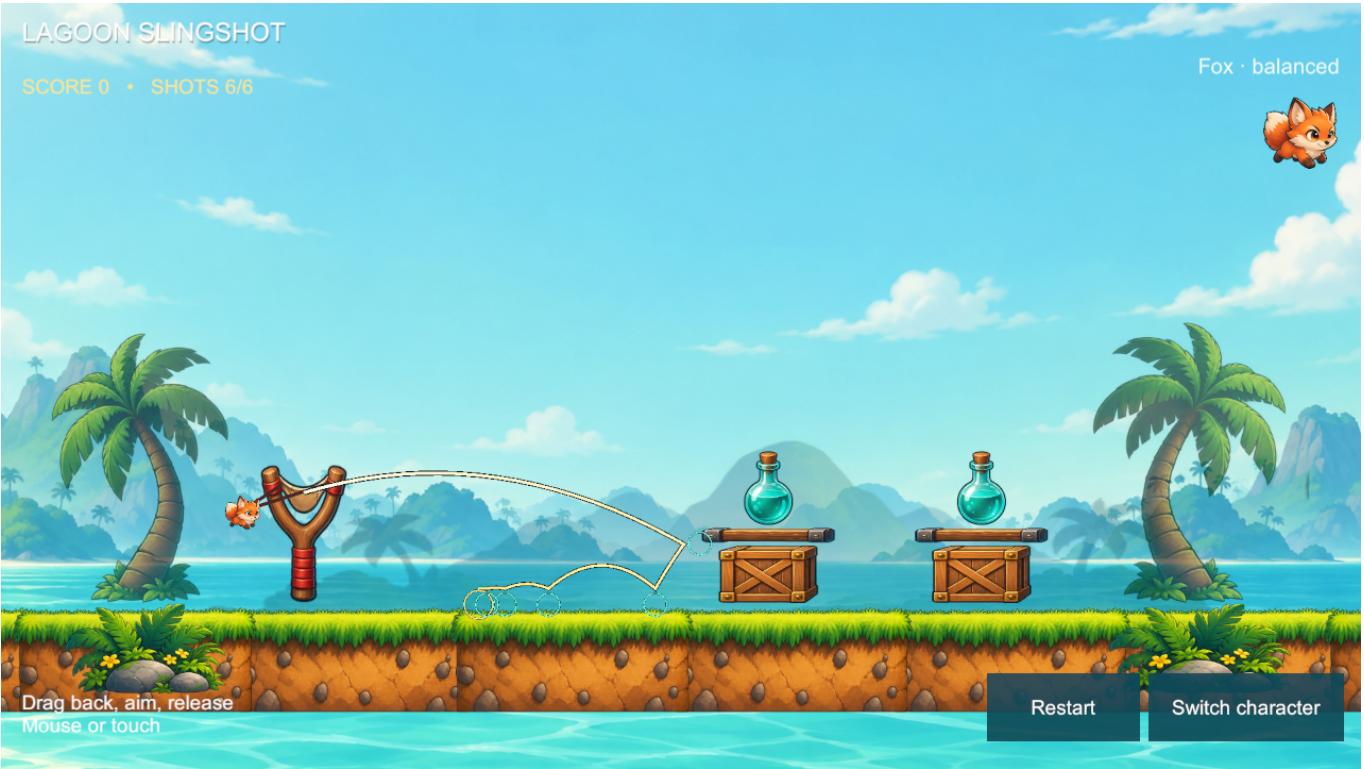
Projectile Toolkit Pro

Beginner Tutorial & Runtime API Guide

Version 0.1.0 | Unity 6000.3.13f1 | C#

Start with one component. Choose a preset. Add only the optional systems your game needs.

TRAJECTORY	PROJECTILE	LAUNCHER
Predict and visualize a path. No spawning required.	Move a projectile and handle its contacts.	Aim, select a prefab and launch it.
ProjectileTrajectory	ProjectileController	ProjectileLauncher



Included 2D Slingshot example. Sample artwork and HUD are optional.

Choose your starting point

Preview only: Tutorial 01. Build a reusable prefab: Tutorial 02. Fire from your own launcher: Tutorials 03-04. Integrate code: Tutorial 09.

This guide describes the current three-component workflow. Pipeline/device compatibility limits are listed in Tutorial 12.

TUTORIAL 01

Your first trajectory

Create a visible path without a projectile prefab, launcher, pool or global manager.

- 01** Open **Tools > ERAGANX > Projectile Toolkit Pro**. Choose **2D** or **3D** under Game type.
- 02** Click **Create Trajectory**. Unity selects the new GameObject. Keep the Inspector in **Basic** mode.
- 03** Use **Self** as the Source. Pick a matching ballistic preset. Position the object where your path should begin.
- 04** Set Direction and Speed. For a 2D arc, try direction **(1, 0.65, 0)** and speed **14**. For 3D, try **(0, 0.65, 1)**.
- 05** Enable trajectory visualization and choose a Path style. Start with **Line** before adding animation.
- 06** Set **Preview distance** to 30 and **Preview duration** to 3 seconds. Enter Play Mode with a camera viewing the path.

Expected result

A visible path begins at the object. Nothing is spawned. A floor with a matching 2D/3D collider lets collision prediction find a contact.

Connect an external source later

Source	Use it when...
Self	The trajectory component supplies its own starting Transform and motion settings.
Assigned Transform	The path follows another object; it does not need the same parent.
Assigned Launcher	The path reads the selected projectile, muzzle, aim and power from a launcher.
Custom provider	Your runtime component implements ITrajectoryDataProvider.

Scene View preview is a setup aid. Check the Game View camera, material and layers when validating the actual rendered path.

TUTORIAL 02

Create a projectile prefab

The prefab is the source of projectile movement, forces, collision rules and projectile effects.

- 01** In the Manager, select the correct game dimension and click **Create Projectile**. Begin with a simple Ballistic or Straight preset.
- 02** In Basic, adjust Speed, Gravity and Collision. To test a scene object immediately, enable **Launch on Play** and set Launch direction.
- 03** Put your model or SpriteRenderer on a **child** object. Assign that child as **Visual root** in Advanced > Visuals when using a visual profile.
- 04** Use a root collider/body matching the chosen dimension. Start with a circular 2D or spherical 3D collision shape. Keep the launch origin outside solid geometry.
- 05** For a launcher-controlled prefab, turn **Launch on Play OFF**. Drag the configured root from the Hierarchy into your Project folder to save a prefab.

Example hierarchy	Purpose
Ball (root)	Projectile Controller plus the matching movement/physics setup.
Visual (child)	SpriteRenderer or mesh. This child can spin, align or scale independently.
Optional effects (children)	Authored trails, particles or animation objects.

Create a ball and a missile

Save a Ball prefab using Ballistic movement, gravity and a bounce response. Save a Missile prefab using Homing movement and its own speed/turn settings. Give each its own visual child. A launcher can switch between them without repeating their movement settings.

One movement authority

Do not move the root Transform from another script while the toolkit controls movement. Rigidbody mode uses Unity physics; swept/kinematic modes use the toolkit movement simulation. Do not add conflicting 2D and 3D bodies.

Switching the preset on an already authored body does not convert every custom collider setup. Review the body and collider when changing dimensions.

TUTORIAL 03

Connect a launcher

A launcher owns where and how projectiles are fired. The projectile prefab still owns its physical behavior.

- 01 Choose the matching game dimension and click **Create Launcher**. Assign your projectile prefab in its Basic Inspector.
- 02 Create a child Transform named **Launch Point**. Place it at the barrel or hand exit and assign it to the launcher. An empty reference uses the launcher Transform.
- 03 Choose **Mouse / touch** for click/tap firing, **Drag to aim** for pull-and-release input, or custom/script input for your own controls.
- 04 Set Launch power and Cooldown. Supply an Aim camera if the scene does not have a correctly tagged MainCamera.
- 05 Under **Optional features**, choose **Add trajectory preview**. The helper connects a Projectile Trajectory for you. You can also assign a trajectory on a different GameObject.
- 06 Enter Play Mode. Aim and fire. Confirm the first preview point and the spawned projectile share the same launch position.

Ballistic launch example

With prefab speed 12 and launch power 1.25, initial speed is 15 units/second, subject to the launcher power range. Gravity and wind then affect flight according to the projectile mode.

Where each setting belongs

Change this...	Edit this component
Movement, speed, gravity, wind, physical collision response	Projectile Controller on the prefab
Aim, power, cooldown, loadout and pooling	Projectile Launcher
Visible preview distance/time, path style and animation	Projectile Trajectory
Intentional per-entry variation	Loadout Overrides, only when needed

If a custom input adapter is assigned, it takes precedence over automatic aim input. Remove or replace it deliberately when changing input ownership.

TUTORIAL 04

One launcher, multiple prefabs

Use a loadout to switch between your ball, missile and other projectile prefabs at the same launch point.

- 01 In the Project window, use **Create > ERAGANX > Projectile Toolkit Pro > Projectile Loadout**.
- 02 Add one entry per prefab. Give each entry a unique Id, readable Display Name and optional Icon. Assign the prefab.
- 03 Leave the optional movement, collision and animation overrides unassigned unless you need a deliberate variation. The prefab supplies the defaults.
- 04 Assign the loadout to Projectile Launcher under Advanced > Launch configuration. Select **Active Projectile Index**: 0 is the first entry, 1 is the second.
- 05 Change the index in Play Mode, or call SelectProjectile from code. The launcher applies the selected prefab and effective movement rules before the next launch/preview.

Entry	Prefab defaults	Optional override
ball	Ballistic movement, ball speed, gravity and bounce	A slower training-ball speed
missile	Homing movement, missile speed and visual profile	A longer per-entry cooldown

No duplicate data required

Do not copy every prefab value into its loadout entry. Enable an override only for the values you intend to change. Local instance edits preserve shared preset assets; editing a shared asset itself affects its users.

Selection behavior

- Invalid entries or indices block launching instead of silently firing the previous type.
- Switching selection cancels a pending burst; the existing cooldown remains.
- A linked trajectory reads the selected launcher rules. Its own preview distance and animation settings remain independent.

```
launcher.SelectProjectile(1);           // second entry
launcher.SelectProjectile("missile");   // unique entry Id
launcher.LaunchSelected();              // respects cooldown
```

TUTORIAL 05

Design the preview

Change how much of the path is visible without changing the projectile's actual travel range.

Length, time and bounce count

Control	Meaning
Preview distance	Maximum visible path length in world units, not a straight-line radius.
Preview duration	Maximum simulated time shown by this preview.
Max preview steps / buffer	Calculation and point budgets. Too small a budget can truncate a path.
Preview Bounce Limit = 0	Show the initial arc up to the first bouncing contact.
Preview Bounce Limit = N	Allow up to N reflected arcs after the initial arc.
Preview Bounce Limit = -1	Follow the other time/distance/physics bounds.

Projectile lifetime, actual travel range, collision budgets and surface responses can end a preview earlier. A request for ten bounces does not guarantee ten visible bounces.

Style, animation and markers

Choose Line, Dots, Dashes, Arrows or an animated style. In **Advanced > Path animation**, adjust animation speed, direction, reveal and fade. The visual profile supplies geometry, color, width, materials and contact markers.

Enable impact/bounce/landing markers in the visual profile. Surface rings are flat contact markers, not projected decals. Use a suitable surface offset for decorative geometry; foreground objects can still occlude them.

Add wind

For a launched projectile, set **Projectile Controller > Advanced > Movement > Wind** on its prefab. A launcher-linked preview inherits it. For an independent trajectory, use **Advanced > Standalone motion > Wind**.

Wind is acceleration

(3, 0, 0) pushes toward world +X in units/second squared. It is not WindZone wind speed. With no drag, its additional displacement is 0.5 x wind x time squared. The gravity-only high/low arc solver does not compensate for wind or drag.

TUTORIAL 06

Movement, collision and targets

Start with shared-kernel movement when you want configurable preview and collision responses to follow the same rules.

Movement family	Setup / prediction notes
Straight / Ballistic / Kinematic	Straight motion or force-based travel with swept circle/sphere queries.
Homing / Returning	Supply a target where required. Moving-target prediction uses a snapshot approximation.
Bezier / Spline / FollowPath	Assign authored controls/path points. Contacts terminate prescribed positional motion.
Orbit / Spiral	Use a fixed center and configured orbit settings. These are authored position modes.
Rigidbody	Unity owns physics integration. Preview is approximate; use native collider materials and layers.

Make a bouncing ball

- 01 Use the Bouncing preset or select a swept movement mode with Collision enabled.
- 02 In Advanced > Collision, set Response to Bounce. Set Restitution, Friction, collision layers and contact budgets.
- 03 Add a matching floor collider. Optional Projectile Surface profiles let different surfaces use different responses.
- 04 Configure the preview bounce limit on Projectile Trajectory separately from the projectile's real contact budget.

Homing and returning

For Homing, assign Homing target and adjust turn speed and target lead. For Returning, enable **Return Via Target** to visit the target before returning to the original launch position. Without a target, it falls back to the configured return timer. Collision, lifetime and range still apply.

Native physics differs

In Rigidbody mode, actual Unity physics materials control restitution/friction. Swept surface coefficients do not replace the native solver. Native pass-through requires suitable trigger shapes. Do not expect exact prediction of arbitrary dynamic collisions or external scripts.

TUTORIAL 07

Mouse, touch and drag aiming

Choose one input owner. Use the launcher's built-in input for a quick prototype, or an explicit adapter for your game UI.

Ordinary click/tap aiming

Mouse / touch input converts a screen position onto an aiming plane and fires on click/tap. Set the correct Aim camera and plane for your game. 2D normally uses an XY plane; a top-down 3D setup can use a horizontal plane.

In the 2D Playground sample, normal clicks inside the cannon's swept muzzle reach preserve the previous aim and do not fire. Click outside the barrel reach to aim and shoot. This is a sample safety policy, not a restriction on LaunchAt.

Drag to aim

- 01** Enable Drag to aim. Use Reverse Drag for a slingshot: pull backward to launch forward.
- 02** Set Minimum Drag Distance, Maximum Drag Distance and Drag Power Scale. The launcher also clamps power to its allowed range.
- 03** Press, pull and hold to aim. A stationary hold stays stable. Release a valid gesture to queue one shot; a click or sub-threshold drag does not fire.
- 04** For a cable/pouch visual, read **CurrentDragOffset**. This is the physical bounded pull, not CurrentAimPoint, which can be a distant direction reference.

Slingshot example

The supplied scene matches a 2.4-unit input drag limit and 2.4-unit Maximum Pull visual limit. Half pull stretches the pouch by 1.2 units. Extra drag cannot stretch it beyond 2.4. Release/cancel returns it to rest.

Canvas and gamepad integration

CanvasAimArea supplies pointer/touch drag events to PointerAimInput. Disable the polling component when that explicit UI area owns the same gesture, avoiding duplicate input. Optional Input System integration is supplied separately; the toolkit does not require it for every workflow.

SetGamepadAim supports a custom gamepad adapter. SetInput accepts your own IProjectileInput. CancelAim cancels a gesture, and ResetAim clears the adapter's retained aim state.

TUTORIAL 08

Projectile visuals and effects

Keep presentation on a visual child so animation does not change the root collision shape or fight physics.

- 01** Create or select a Projectile Visual Profile. Assign it to the Projectile Controller under Advanced > Visuals.
- 02** Assign the child Transform as Visual root. Choose the forward axis that matches your sprite or model.
- 03** Enable alignment to velocity and adjust rotation smoothing. Add spin, wobble, scale or color-over-lifetime curves as needed.
- 04** Assign optional sprite frames or an Animator. Configure supported Speed (float) and Launch (trigger) parameters where your animation uses them.
- 05** Add authored child TrailRenderers or optional particle prefabs for spawn, movement, collision, bounce and despawn. Test reuse through the launcher's pool.

Effect type	When it is used
Trajectory marker	Shows a predicted future contact. It does not trigger gameplay impacts.
Collision / bounce particles	Play when an actual projectile contact occurs.
Trail / moving effect	Follows the active projectile and is reset for pooled reuse.
Persistent scorch / decal	Implement through your game's contact callback and rendering pipeline.

Materials and rendering

Trajectory custom materials should support vertex color and UVs. Scrolling textures require a compatible texture-transform property such as `_MainTex_ST`. Start with the included path material before substituting a custom shader.

Check the target pipeline

Basic visuals do not require post-processing or VFX Graph. There is no dedicated VFX Graph adapter. Check materials in your actual Built-in, URP or HDRP configuration; complete cross-pipeline compatibility is not verified for this release candidate.

TUTORIAL 09

Integrate your own scripts

Use the public components directly. The example assumes a configured launcher/prefab, and a Projectile Trajectory reference assigned in the Inspector.

```
using UnityEngine;
using ERAGANX.ProjectileToolkit;

public sealed class ExampleWeapon : MonoBehaviour
{
    public ProjectileLauncher launcher;
    public ProjectileTrajectory preview;
    public Transform target;

    void Start()
    {
        // This script owns input; disable automatic polling.
        launcher.SetInput(null);
        preview.SetSource(launcher); // may be on another object
        preview.SetPreviewDistance(30f);
        preview.SetPreviewDuration(3f);
        preview.SetPreviewBounceLimit(2);
        preview.SetAnimationSpeed(1.5f);
        preview.SetVisible(true);
    }

    void LateUpdate()
    {
        if (!target) return;
        launcher.aimPoint = target.position;
        // Automatic refresh can remain enabled for this setup.
    }

    // Call these from your UI or existing input system.
    public void Fire()
    {
        if (target) launcher.LaunchAt(target.position, 1f);
    }

    public void SelectMissile()
    {
        if (launcher.SelectProjectile("missile"))
            preview.Refresh();
    }
}
```

Create the script after importing the package, then assign references. The loadout must contain a unique entry Id of missile. LaunchAt returns false when validation, cooldown or gameplay conditions block firing.

For a projectile without a launcher: disable Launch on Play, then call **controller.Launch(origin, velocity)**. Pause(), Resume(), Stop(), SetTarget(target) and ResetProjectile() are also available. Do not call Tick yourself unless you intentionally use manual simulation.

TUTORIAL 10

Pooling and practical budgets

Reuse resources first, then measure the game on its target device. Pooling trades startup work and memory for reduced creation work during play.

- 01** Enable **Reuse projectiles** / automatic pooling on the launcher. The ordinary setup does not require one manually added pool per prefab.
- 02** For loadouts that must be ready immediately, enable **Prewarm all loadout prefabs** before startup. It is off by default. Only valid, enabled toolkit pools are prepared.
- 03** Use per-entry pooling overrides only when a projectile needs a different capacity. The first configuration for an existing prefab pool wins while it exists.
- 04** Hide or disable previews that are not needed. Set preview refresh frequency, distance, duration, step count and geometry budgets for your game.
- 05** Profile a representative simultaneous-projectile load on your target hardware. Check frame time, allocations, physics cost and rendering cost separately.

Prediction scheduling

Projectile Trajectory uses reusable buffers and a scheduler. Automatic refresh is frequency bounded. Cache age lets an unchanged source periodically refresh when scene geometry changes. Changing buffer capacity replaces the buffer at the next allowed calculation.

If a performance profile is assigned, it owns the indicated quality/refresh/buffer/cache/geometry budgets. Clear it before editing those local controls. For manual changes, use `Invalidate()` or `Refresh()`; do not force several identical refreshes every frame.

Events and object ownership

Projectile events include Launched, Collided, Bounced, Stopped, Expired and Despawnd, with C# callbacks and `Inspector UnityEvents`. Subscribe/unsubscribe with your component lifecycle. Contacts provide the impact point and normal for your gameplay integration.

Toolkit versus ordinary prefabs

A compatible ordinary prefab can use a root `IProjectileLaunchReceiver` or a dynamic native `Rigidbody/Rigidbody2D`. It owns its movement, contacts and cleanup; toolkit typed pooling/events do not automatically apply. Its preview is approximate.

No FPS, allocation-free-everywhere or Android/iOS performance guarantee is made by this guide. Resource-budget presets are starting points, not device benchmarks.

TUTORIAL 11

Explore the supplied samples

Use the Manager's Samples section or open the scene asset directly. Work in a copy before modifying a sample for your own game.

Scene	What to explore
Trajectory Basics	Independent trajectory previews without spawning projectiles.
2D Projectile Playground	Ammo switching, a fixed cannon mount, click/tap and drag/release aiming.
3D Projectile Playground	3D launching, trajectory previews and a visible cannon.
Collision and Bounce Showcase	Separate response stations for bounce, slide, stop and attachment.
Homing and Targeting	Guidance, target selection and moving-target behavior.
Launcher and Aiming	Power, direction and optional drag/release launcher controls.
2D Slingshot Demo	Mouse/touch pull-and-release, bounded cable stretch, character switching and Restart.
3D Arrow Demo	Arrow launch, elevation/yaw, wind and scoring targets.
3D Cannon Demo	Existing first-person bomb-throwing example. Not included in the latest focused verification.

Essential presets

The public preset collection contains Straight, Ballistic, Bouncing and Homing configurations for 2D and 3D, three trajectory styles (Line, Flowing Dots and Arrows), a Default Launcher, and Balanced / Mobile Low resource budgets: 14 public assets in total.

Additional special configurations belong to the samples. They demonstrate existing movement capabilities without filling the main preset browser with variations. Sample controls are demonstration code and can be replaced by your own UI.

Avoid broad regeneration

Existing scene-generation utilities are authoring tools. Do not regenerate over a customized sample to solve a local setup issue. Make focused edits or duplicate the sample first.

TUTORIAL 12

Troubleshooting and limits

Use this checklist before adding extra components or copying the same settings into several places.

Symptom	Check
No path is visible	Enable visualization. Check camera view/layers, a nonzero direction/speed and the trajectory source. Start with a simple Line material.
The path ends too soon	Preview distance/time/steps/buffer, projectile lifetime/range, contact limits and stopping surfaces can all shorten it.
A setting appears locked	Check its feature toggle, assigned source or performance profile. Edit physical rules on the prefab; preview length/animation on Trajectory.
The wrong prefab fires	Check the loadout, zero-based index and valid prefab reference. Confirm no custom script continually restores the previous index.
The projectile fires by itself	Disable Launch on Play on launcher-controlled prefabs. Check automatic input versus your custom input adapter.
Collision or bounce differs	Check matching dimension, collider layers, triggers and movement authority. Rigidbody mode uses native materials; sweeps use toolkit responses.
Markers disappear into surfaces	Adjust visual marker surface offset. Rings are flat meshes; they do not project over curved terrain.
Cable jumps to full stretch	Use CurrentDragOffset for the pouch, not CurrentAimPoint. Match the intended visual/input pull limits.

Prediction boundaries

Shared sweeps model circles/spheres with fixed-step chords. Native Rigidbody previews are approximate. Isolated native preview supports explicitly supplied static box proxies, not a complete replica of arbitrary gameplay. Moving targets are extrapolated snapshots; future moving-platform geometry is not simulated.

Authored positional paths do not physically bounce while continuing their original curve. Custom motion must explicitly support prediction. Universal prediction of arbitrary MonoBehaviours, animation-driven objects or external physics changes is unsupported.

Before shipping your game

Validate the target rendering pipeline, real input devices, resolution/aspect ratios and representative performance. Unity 6000.3.13f1 is the development Editor; physical Android/iOS tests and complete URP/HDRP acceptance are not claimed.