SolidCore Coroutines for Unity

Zero-allocation, struct-based coroutines that tick thousands of sequences per frame with no GC.

SolidCore Coroutines is a drop-in alternative to Unity's `StartCoroutine` built for scale. Where Unity allocates an iterator state machine (plus a `WaitForSeconds` object) for *every* coroutine, SolidCore stores coroutine state in contiguous native memory and drives it from a single explicit `Tick`. The result: allocation is flat regardless of how many coroutines are alive — hundreds of thousands of concurrent sequences run with near-zero GC per frame and a rock-steady frame rate.

- **Zero per-frame GC** — coroutine state lives in native buffers, not the managed heap.
- **Batch spawning** — start an entire volley of N coroutines from one reusable template in a single allocation-free call.
- **Single- or multi-threaded** — a single-threaded `CoroutineRunner` for the game loop, or a multi-threaded `CoroutineJobSystem` that shards work across worker threads.
- **Ordinary C# delegates** — `Do(Action)` and `WaitUntil(Func<bool>)` take normal managed callbacks. Fully **IL2CPP / Mono** compatible (no unmanaged function pointers).
- **Deterministic** — you own the tick, so execution is explicit and testable.

- **Live diagnostics** — O(1) active count, per-tick wall-clock cost, throughput, and parallel-speedup metrics.

What's in this package

This asset contains two things:

1. **The runtime library** — precompiled assemblies bundled directly in the asset:

```
Assets/Solid24 Systems/Runtime/Plugins/
  SolidCore.Coroutines.dll   ← the scheduler (netstandard2.1)
  SolidCore.Burst.dll        ← native instruction storage
  SolidCore.Collections.dll  ← callback registry backing
```

They're auto-referenced, so any script can `using SolidCore.Coroutines;` with no setup. The `.xml` files next to each DLL provide inline API documentation (IntelliSense/tooltips) in your IDE.

2. **The demo & samples** (`Assets/Solid24 Systems/Samples/`) — two side-by-side benchmark scenes that fire projectile "volleys" and chart the difference between Unity coroutines and SolidCore on a live performance HUD (GC/frame, worst frame, FPS, tick time). See the Getting Started guide.

Requirements

Unity	6000.0 (Unity 6) or newer
Scripting backend	Mono **or** IL2CPP
Render pipeline	Runtime is render-pipeline agnostic. **The sample scenes require URP** — in Built-in RP the demo materials render magenta (see the Getting Started guide).
Package dependencies	The demo scenes use TextMeshPro for the HUD. The runtime library is standalone (bundles its own `SolidCore.Burst` and `SolidCore.Collections`).

Quick start

```
using SolidCore.Coroutines;

// 1. Create a runner, pre-sized for the peak number of concurrent coroutines.
using var runner = CoroutineRunner.Create(expectedPeak: 10_000);

// 2. Build a reusable template ONCE (no allocations at runtime).
InstructionSequence seq = runner.Sequence()
    .Do(() => Debug.Log("spawn"))
    .WaitForSeconds(1.5f)
    .Do(() => Debug.Log("despawn"))
    .Build();

// 3. Spawn a whole wave from the template in one batch call.
runner.Start(seq, count: 10_000);
```



```
// 4. Advance one frame in your game loop.
void Update() => runner.Tick(Time.deltaTime);
```

Full walkthrough — including running the demo, integrating into a `MonoBehaviour`, and the multi-threaded job system — is in **GettingStarted.md**.

Complete type/method listing is in **API_Reference.md**.

Documentation

File	Contents
GettingStarted.md	Install, run the demo, and integrate SolidCore into your own project.
API_Reference.md	Every public type and member, grouped by role.
CHANGELOG.md	Version history.
Support.md	Support channels, EULA, and known limitations.

Support

- **Email:** Support@Solid24Systems.com
- **Website:** https://solid24systems.com

See Support.md for details.
