AnySync Documentation

Links

Online Documentation
Introduction Video
Forum
Asset Store
Support Email

Introduction

AnySync is an interpolation library designed to help with synchronizing motion over the network. Developers need to use their own network code to utilize it. Netcode could be built using UNet, Photon or any other networking library, including custom ones built from scratch.

The idea behind this asset is to push positions into a buffer and interpolate through them to create accurate keyframed motion. Core features include the use of extrapolation for hiding lag spikes and compensating latency.
There are 4 example scenes made to soften up the learning curve. They are split up to 3 difficulty levels, explaining different techniques to approach the problems for both RPC and serialization-based netcode design. Examples use UNet and PUN2.

AnySync is made for programmers, so it's not the easiest asset to use. If you struggle with coding and expect a simple drag and drop setup, this might not be the right tool for you.

Examples

Check out the introduction video linked above, then move on to example projects.

UNet examples (RPC-based approach).

- Basic code using Rigidbody2D.
- Advanced code using CharacterController.
- Expert code using Rigidbody.

Instructions:

1. Open one of the example scenes.
2. Make a standalone build and run it.
3. Press the play button in your editor and start a host.
4. Start client in your standalone build.

Photon PUN2 example (Serialization-based approach).

- Advanced code using CharacterController.

Instructions:

1. Make sure you have PUN2 imported and properly set up in your project.
2. Import the PUN2 example package.
3. Open the example scene.
4. Make a standalone build and run it.
5. Press the play button in Unity editor.

Integration

Best way to learn how to integrate AnySync is to go through the examples, but here's a quick rundown just in case.

1. Include AnySync namespace.

```
using AnySync;
```

2. Create a new instance of MotionGenerator.

```
private MotionGenerator _motionGenerator = new MotionGenerator();
```

3. Send position data through network and feed new keyframes to MotionGenerator.

```
private float _timeSinceLastSync;
private void SendSyncMessage()
{
	CmdSync(_timeSinceLastSync, transform.position);
	_timeSinceLastSync = 0f;
}

[Command]
private void CmdSync(float interpolationTime, Vector3 position)
{
	_motionGenerator.AddKeyframe(interpolationTime, position);
}
```

4. Progress playback and apply new position data.

```
private void Update()
{
	_timeSinceLastSync += Time.deltaTime;
	if (_motionGenerator.HasKeyframes)
	{
		_motionGenerator.UpdatePlayback(Time.deltaTime);
		transform.position = _motionGenerator.Position;
	}
}
```

Settings

MotionGenerator can be configured the way you want. Default settings will suit most projects, so don't bother changing anything while integrating it for the first time.

Interpolation Latency

- Controls buffer length and extrapolation power, could be set in MotionGenerator constructor. It's the only option likely to be changed for your project.
- Default value is 0.12ms. It assumes a send rate of 10 messages per second, which equals to 0.1ms send interval plus 0.02ms for frame time and expected packet jitter.
- Values way over send interval increase resistance to bad network conditions at the cost of increased movement delay.
- Values below send interval compensate latency by using extrapolation, but it could introduce overshooting issues (clipping through objects, floor, walls) and rubber-banding during abrupt velocity changes. It's possible to set a negative value for extrapolating far into the future.

Error Correction Speed

- This setting controls position error correction speed caused by extrapolation. It's using smooth lerp algorithm which is barely noticeable on small errors, but ensures fast correction if severe errors occur.
- Default value is 10. It's not too harsh, but gets the job done in time. Recommended to leave the default value.

Time Correction Speed

- MotionGenerator is trying to stay InterpolationLatency late from the present time. Sometimes network messages are lost altogether, or delivered earlier/later than expected - this is where time drift occurs.
- Default value is 3. This ensures balanced behavior that works well under packet jitter and packet loss. Recommended to leave it as is.

Disable Extrapolation

- Sometimes you can't use extrapolation. Moving objects will never overshoot, but you lose all extrapolation benefits. Make sure to use high enough Interpolation Latency to avoid issues, at least higher than send interval.
- By default, extrapolation is enabled. Recommended not to touch this setting unless there is no other way around it.

Advanced features

It's worth noting that nearly everything in MotionGenerator is public, protected or virtual to allow users to extend the functionality by making a custom class that inherits from it.

Executing code in sync with the movement.

- InvocationSync class is an extension that helps to execute some code at the right time with the movement. See UNetRigidbodyColorChanger in Expert example to learn how to use it. Find the code link in examples page.

Skipping sync while idle

- To save bandwidth and fit within tight network limitations, you can cease sending messages while object is not moving. Before going idle, last keyframe must set the object velocity to 0. Expert example is synchronizing velocity, so it happens automatically.
- But if you don't sync velocity over the network, MotionGenerator calculates velocity using last 2 keyframes, so you have to duplicate last keyframe to nullify the velocity. It's a little bit tricky, see how it's handled in Advanced UNet and Photon examples.

Teleporting

- Sending a keyframe with 0 interpolation time triggers teleportation. See how it's handled in Advanced UNet and Photon examples. The latter is a bit tricky because Photon can only send sync messages in a certain time interval.

Improving extrapolation accuracy

- The easiest way to significantly improve extrapolation accuracy is to provide velocity with every keyframe, just like in Expert example.
- Rubber-banding effect could be mitigated by adding inertia and smooth acceleration to character movement. Extrapolation doesn't like abrupt velocity changes.
- Clipping through objects due to overshooting could be mitigated by using Rigidbody.SweepTest or similar techniques.
- Simply reducing velocity helps a lot as well.

Frequently asked questions

If you are having problems, please attach a playable reproduction project when contacting support. It will help to solve your issue quickly and efficiently. A support ticket with just some code chunks will most likely be a waste of time.

Weird or laggy movement.

- Check if network messages are filtered properly. Sometimes clients receive messages they were not supposed to.
- MotionGenerator position and rotation data should be applied to Transform, not the Rigidbody. Physics and Update loops run in separate threads, so mixing those might cause jitter. Make sure you use interpolation in rigidbody settings as well.
- If you don't want to specify velocity in the AddKeyframe method, use an overload without velocity instead of hardcoding Vector3.zero.
- Make sure you don't send keyframes with 0 interpolation time on accident. Those are used for teleportation (they ignore smooth error correction).

Rubber-banding movement (object moves back when stopping).

- Try setting InterpolationLatency to be 0.02 seconds higher than your send interval, or use a default MotionGenerator constructor with no settings if not sure.

Object is drifting while supposed to stay still.

- See "Skipping sync while idle" in "Advanced features" page above.

Can I really use it with my own networking solution ?

- Yes, you can. It's completely netcode-independent.

My custom server doesn't use Unity. Do I need to run AnySync on it ?

- No, you don't. UNet examples use MotionGenerator in [Command] methods only for tutorial visualization purposes. Custom server should only send or relay position messages to clients.

I'm not happy with the asset. Can I refund it ?

- Refunds are available within 6 months after purchase. No questions asked. (after a refund you will get no further support and miss out on updates)
- Send an email stating that you wish to request a request a refund to bananapartydev@gmail.com with the invoice PDF file attached.

Change log

V2.02

- Updated Expert example to not mislead people into using rigidbody.position instead of transform.position.
- Cleared up confusion caused by rigidbody.angularVelocity returning radians, while AddKeyframe expects degrees.

V2.01

- Updated PUN2 example to support all observe options in PhotonView - whether it’s Unreliable, Unreliable On Change or Reliable Delta Compressed. Sample code is now way easier to understand and more convenient to use with other networked components together.

V2.0

- Major update that includes some breaking changes. Backup your project and prepare to update your code in order to upgrade.
- Naming changes. SyncBuffer was renamed to MotionGenerator, TargetLatency renamed to InterpolationLatency.
- SyncBuffer (MotionGenerator) is now completely separate from MonoBehaviour. Refer to “Integration” page and new examples to update your code.
- Added InvocationSync extension. Now it’s possible to execute RPCs in perfect sync with the movement.
- ECS compatibility. It's not exactly designed for pure ECS since data is not separated from the behavior code, but it should work fine regardless if you put MotionGenerator into a component struct.
- Removed acceleration from the AddKeyframe method. It was making the code a bit daunting and nobody used it.
- All examples are completely remade from scratch. Old PUN (v1) and Websocket examples were removed.
- AddKeyframe parameter order was changed.
- PlaybackTime getter is now public, so it’s possible to use it to manually compensate for time drift from losing packets.
- Lots of cosmetic changes to make the code less daunting.

V1.07

- Remade PUN 2 example to use CharacterController.

v1.06

- Added PUN 2 example.

v1.05

- Added simple 2D UNet example to help with the learning curve.
- Fixed issues in Photon example when PhotonView has multiple observed components.
- Removed all inspector settings except TargetLatency to avoid confusion. Everything is still accessible from the code.
- Allowed greater flexibility by using protected fields and virtual methods.

v1.04

- Restricted TargetLatency inspector range to 0.0 - 0.5 in order to avoid confusion. Use scripts to assign a negative value in the future (if you really need to).
- Fixed UNet and WebSocket examples to send 10 messages per second instead of 8.

v1.03

- Added WebSocket example.
- Fixed more issues in Photon example.
- Fixed teleporting while object is idle.

v1.02

- New documentation.
- Fixed some potential issues on teleporting in Photon example.

v1.01

- Added an example package for Photon Unity Networking.