Simple Network Sync Documentation

Simple Sync Animator
Simple Sync Transform

Current version of this document online here

Contact the author at: davincarten@gmail.com

I am also regularly in #Unity-Dev on Discord with the screenname **emotitron** if you have questions.
https://discord.gg/tGDUxp9

Unity 2019+ users:
Be sure a network library is installed before installing Simple Network Sync. UNet is no longer included, and if there is no library (PUN2, Mirror or manually added UNet) you will get a wall of errors.

Documentation is intentionally still minimalistic. PLEASE, contact me with any questions or for help, I am very quick to respond. I am avoiding too much documentation work until the first version of this has finished being updated as use cases come up, as it will quickly become outdated and wrong.

This software is always being improved and tightened up, and the main driver outside of my own usage is the devs using it requesting changes, improvements and fixes. email me or even better hop on to discord and let me know what you think it needs, or let me know how you were able to break it.

Concept

My aim is to create “just works” drag and drop networking components that bring better practices of tick-based simulation syncing and advanced bitpacking to areas of networking most new developers have the most trouble.

The first two components are syncs for Transforms and Animators.

Unlike the built-in UNET NetworkTransform, NetworkAnimator, PhotonTransformView, PhotonAnimatorView as well as other third party transform syncs, this library uses a ring buffer with numbered frames for each net entity (NetworkIdentity/PhotonView). This produces VERY smooth and stable syncs even in adverse network conditions, and also establishes a base for more advanced networking concepts later, like rewind and determinism.

All SimpleNetworkSync components on a networked gameobject serialize into a single byte[] using bitpacking, to give the highest levels of compression possible.

Animator Compression:

	Uncompressed	Network Packed
Normalized Floats	32 bits	10 bits
Other Floats	32 bits	16 bits (half float)
State/Trigger Hashes	32 bits	1-8 bits (based on total #)
Bools/Trigger Params	8 bits	1 bit

SendEveryX value reduces the overall tick rate to a set fraction of the FixedUpdate rate.
Keyframe Rate allows for the use of delta frames, reducing unchanged data elements to 1 bit.

Note about FixedUpdate vs Update timing:

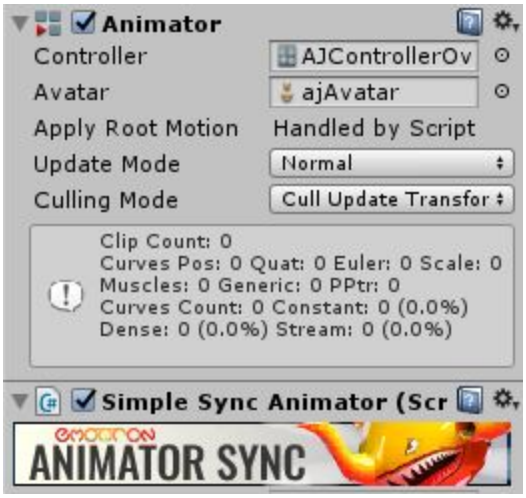
This component set is tuned for games that simulate/move/animate using timings based around **FixedUpdate()**. It will work in simulation free or Update() based simulations as well, but those are generally less than ideal for networking. and are prone to micro-jitter. I always recommend in Unity doing all game logic in fixed, and using Update() for interpolation/extrapolation only when doing networking - rather than applying changes and game logic in Update(). This however is a much larger topic I won’t cover here.

Usage

These components are meant as replacements for NetworkTransform, NetworkAnimator, PhotonTransformView and PhotonAnimatorView. As such, you must do the same ownership checks as you do for those. Specifically **be sure to test hasAuthority (Unet/Mirror) and IsMine (PUN2) in your controllers, and only apply controllers if these are true.** Otherwise, you will be trying to apply inputs to all objects on every connection, even the ones that player does not own.

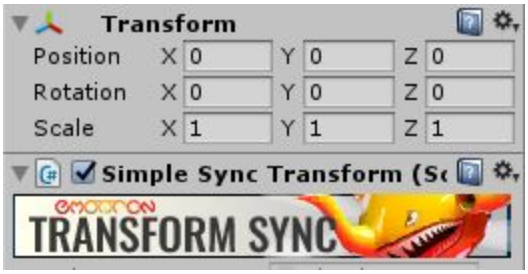
Simple Sync Animator

Place on any GameObject that has an Animator component you would like to sync.



Simple Sync Transform

Place on any GameObject root or child that needs its Transform to be synced.

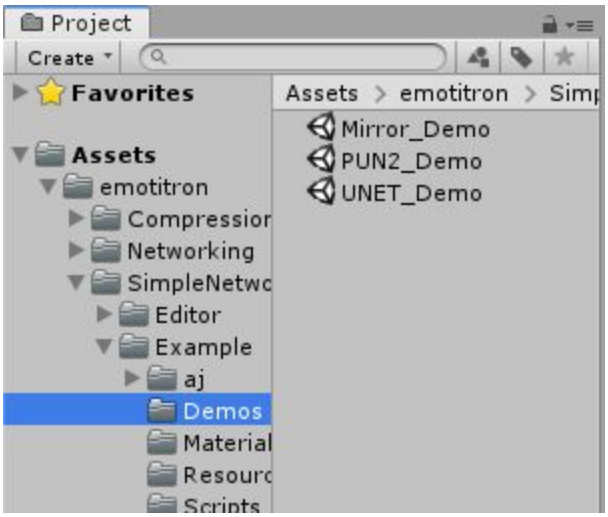


Demos

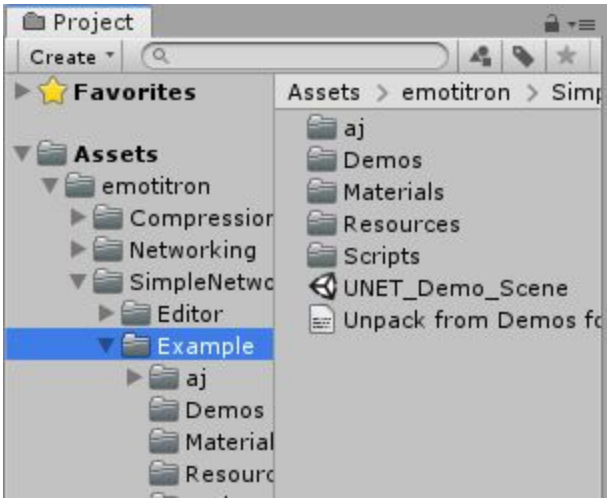
To support the wide range of Network Libraries (Unet/PUN2/Mirror) and range of Unity versions 5.6 to 2019 - it is not possible to keep demo scenes in the project.

To this end the demos are all Asset Packages that need to be opened.

1) Extract the Demo for the Network Library you are using.

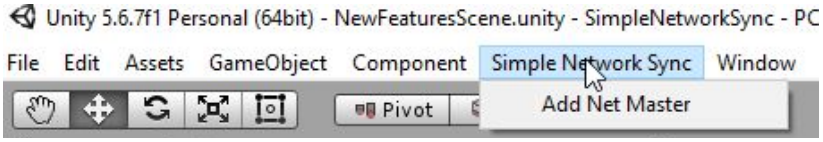


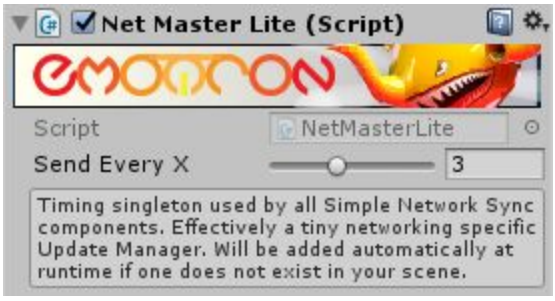
2) Open the newly created scene in the Example folder



Net Master Lite

If you would like to send at a lower rate than every FixedUpdate(), add Net Master to your scene and adjust the Send Every X value to reduce the send rate.





Simple Network Sync Common Fields

Shared fields among sync components.

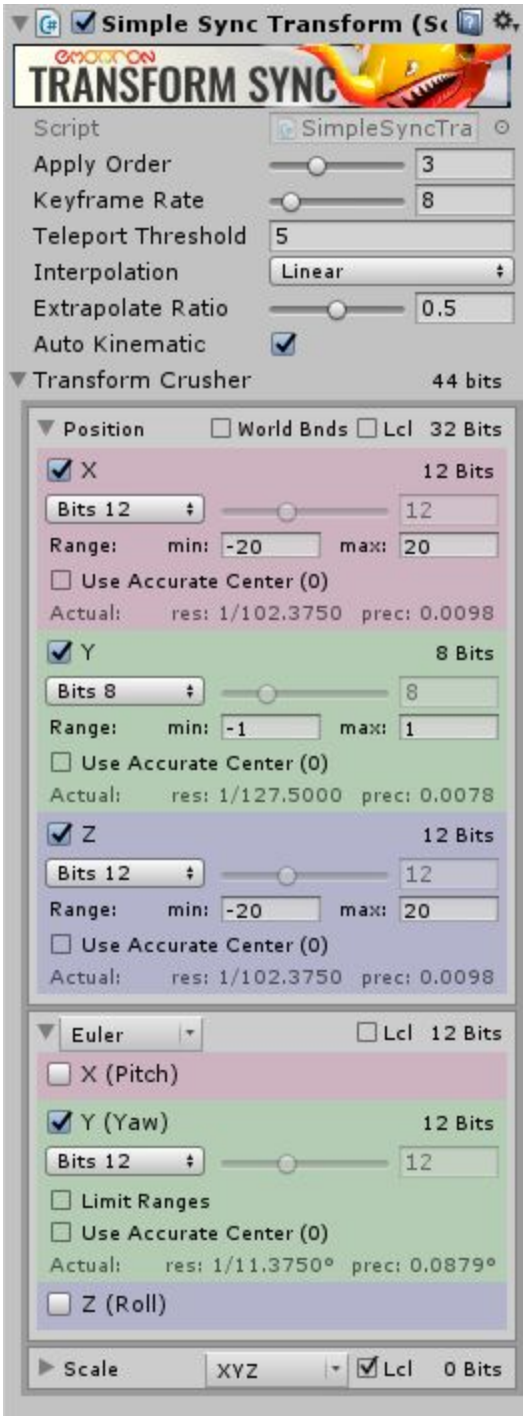
Apply Order
It is recommended that you not change this.
The order in which components on a network object run, in order from lowest to highest. This is used to ensure things that regardless of component order, certain things will always happen first. Some Transform and Animator setup may prefer one or the other to happen first. By default Transform is first, as this produced the best results in my example scene, but it may not be the case with all Animator configurations.
When components have the same Apply Order value, then the order in the gameobject hierarchy is used.

Keyframe Rate
These components are one-directional broadcasts. As such it is possible to reduce data by using keyframes. When no changes occur to elements of synced objects, they will send a 1 bit false flag indicating no content, rather than sending the same data every update. This does mean that loss and late joining players (in the case of PUN2) may be result holding on an out of date value. Keyframes ensure eventual consistency when using delta frames. The greater the value the more data savings, but also a greater risk of odd behavior when packet loss is encountered.

If bandwidth is less of a concern, set this value to 1, and every update will contain a full compressed state.

Simple Sync Transform

The primary NetworkTransform / PhotonTransformView replacement component is SimpleSyncTransform.



Teleport Threshold
The distance in units between frame updates that will trigger a teleport. This exists just to have parity with other transform syncing tools. Teleporting can have different meanings in different games, so ideally you will want to code teleport handling yourself.

Interpolation

- **None** - Objects will snap to the networked states, without lerping.
- **Linear** - Basic Ler[/S;erp are used to interpolate between networked frames. This is likely the mode you want.
- **Catmull Rom** - (Experimental) A more sophisticated lerp that uses 3 points, producing a more naturally curved and accurate path than a basic lerp.

Extrapolate Ratio
Extrapolation occurs when the tick advances, but no frame info has arrived yet on clients for this object. The synced object now has to guess what the next frame's values are. Extrapolation uses the last two values to extrapolate the new frame. The Ratio is the t value used by LerpUnclamped, and acts as a dampener of extrapolation. So for sequential missing frames the extrapolation gets reduced on a curve.
0 = no extrapolation - object will not move on empty buffer.
.5 = evenly dampened - object will lerp less with each tick.
1 = full extrapolation - no damping, will extrapolate until a frame arrives or object is destroyed (disconnect).

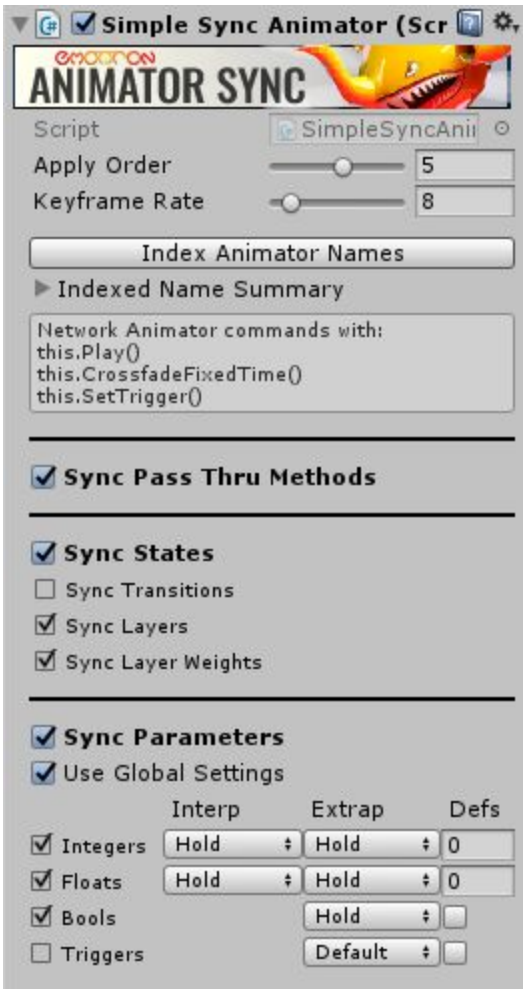
Auto Kinematic
A best guess is made for the handling of owner/server/others on how to set the rigidbody isKinematic. Generally all non-authority instances of the object will be set to isKinematic = true.

Crusher
See TransformCrusher
TRS data (Position, Rotation, Scale) are compressed using my Transform Crusher library, which has its own documentation.

More information about the Element Crusher can be found at in the Transform Crusher documentation.

Simple Sync Animator

The primary NetworkAnimator / PhotonAnimatorView replacement component.



Index Animator Names
One of the primary compression methods of this component is the indexing of all State and Trigger names, so that rather than sending 32 bit hash values for states and ticks every update, MUCH smaller (1-5 bits) indexes are sent instead.

The indexing happens automatically with a lot of hackery behind the scenes, but it doesn't always fire (since it can't be forced during the build process). You may see warnings in the log asking you to press this button. It never hurts to press it, especially after making changes to your Animator Controller.

If at runtime an index isn't found, it will just fall back to sending the full 32bit hash. Things will still work, but your network usage will increase.

Sync Pass Thru Methods
If you call SetTrigger(), Play(), CrossFadeInFixedTime() and such through this component, it will network those calls, and pass along the command locally to the Animator. This often can give the most in sync result, for a relatively low network cost (less than a byte typically).

Sync States
This is the workhorse of an Animator sync. You likely want to leave this checked.

Sync Transitions
This is of questionable utility, but is included for future improvement and for completeness. By default it is off.

Sync Layers
If you are using layers and want them synced, turn this on. Disable if you are not to save some data.

Sync Layer Weights
If you are animating layer weights, you will want this on as well. If you are not animating them however, be sure to turn it off - it adds up to 10 bits of data per layer per tick.

Sync Parameters
You can globally or selectively select which parameters to sync. Interpolation, Extrapolation and default values can be set, but generally you will just want to leave the defaults.

Net Master

This singleton is created at runtime if none exists in the scene. It exists to ensure all networked objects using Simple Network Sync components fire on the same ticks, and is the primary traffic cop.

Send Every X Fixed - SimpleSync uses FixedUpdate as its primary tick, but can produce a network rate that is a subdivision of that rate. For example the default Unity physics fixedDeltaTime is .02 secs (50 ticks per sec). Setting `[Send Every X]` to 2 would result in a net rate of .04 (25 ticks per seconds) - ½ of the physics rate. This effectively cuts generated traffic in half.