Smooth.SmoothSync Class Reference

Sync a Transform or Rigidbody over the network. Uses interpolation and extrapolation. More...

Inherits NetworkBehaviour.

Public Types

enum	**ExtrapolationMode** { **None**, **Limited**, **Unlimited** } Extrapolation type. More...
enum	**WhenToUpdateTransform** { **Update**, **FixedUpdate** } Info to know where to update the Transform. More...

Public Member Functions

void	**addState** (**State** state) Add an incoming state to the stateBuffer on non-owned objects. More...
void	**Awake** () Cache references to components. More...
void	**checkIfOwnerHasChanged** () Checks if the owner has changed on each received **State**. If it has, add a "fake" received **State** to the **State** array with the current Transform so that you can lerp between it and the first **State** from the new owner. More...
void	**clearBuffer** () Clear the state buffer. Must be called on all non-owned objects if it's ownership has changed. More...
void	**CmdTeleport** (Vector3 position, Vector3 rotation, Vector3 scale, float tempOwnerTime) Echoes a teleport **State** from the host to all clients. More...
void	**forceStateSendNextFixedUpdate** () Forces the **State** to be sent on owned objects the next time it goes through FixedUpdate(). More...
override int	**GetNetworkChannel** ()
override float	**GetNetworkSendInterval** ()
Vector3	**getPosition** () Get position of object based on if child or not. More...
Quaternion	**getRotation** () Get rotation of object based on if child or not. More...
Vector3	**getScale** () Get scale of object. More...
void	**OnEnable** () Automatically sends teleport message for this object **OnEnable()**. More...
override void	**OnStartClient** () Register network message handlers on clients. More...

override void	**OnStartServer** () Register network message handlers on server. More...
void	**registerClientHandlers** ()
void	**RpcNonServerOwnedTeleportFromServer** (Vector3 newPosition, Vector3 newRotation, Vector3 newScale)
void	**RpcTeleport** (Vector3 position, Vector3 rotation, Vector3 scale, float tempOwnerTime) Receive teleport **State** on clients and add to **State** array. More...
void	**setPosition** (Vector3 position, bool isTeleporting) Set position of object based on if child or not. More...
void	**setRotation** (Quaternion rotation, bool isTeleporting) Set rotation of object based on if child or not. More...
void	**setScale** (Vector3 scale, bool isTeleporting) Set scale of object. More...
bool	**shouldSendAngularVelocity** () Check if angular velocity has changed enough. More...
bool	**shouldSendPosition** () Check if position has changed enough. More...
bool	**shouldSendRotation** () Check if rotation has changed enough. More...
bool	**shouldSendScale** () Check if scale has changed enough. More...
bool	**shouldSendVelocity** () Check if velocity has changed enough. More...
void	**stopLerping** () Stop updating the States of non-owned objects so that the object can be teleported. More...
void	**teleport** () Deprecated. Use **teleportOwnedObjectFromOwner()** or **teleportAnyObjectFromServer()**. More...
void	**teleportAnyObjectFromServer** (Vector3 newPosition, Quaternion newRotation, Vector3 newScale) Teleport the object, the transform will not be interpolated on non-owners. More...
void	**teleportOwnedObjectFromOwner** () Teleport the object, the transform will not be interpolated on non-owners. More...
delegate bool	**validateStateDelegate** (**State** receivedState, **State** latestVerifiedState) Validation delegate More...

Static Public Member Functions

static bool	**validateState** (**State** latestReceivedState, **State** latestValidatedState) Validation method More...

Public Attributes

SmoothSync []	**childObjectSmoothSyncs** = new **SmoothSync**[0] Reference to child objects so you can compare to syncIndex. More...
GameObject	**childObjectToSync** Child object to sync More...
float	**extrapolationDistanceLimit** = 20.0f How much distance into the future a non-owned object is allowed to extrapolate. More...
ExtrapolationMode	**extrapolationMode** = ExtrapolationMode.Limited The amount of extrapolation used. More...
float	**extrapolationTimeLimit** = 5.0f How much time into the future a non-owned object is allowed to extrapolate. More...
bool	**forceStateSend** = false Gets set to true in order to force the **State** to be sent next frame on owners. More...
bool	**hasRigidbody** = false Does this game object have a Rigidbody component? More...
bool	**hasRigidbody2D** = false Does this game object have a Rigidbody2D component? More...
float	**interpolationBackTime** = .1f How much time in the past non-owned objects should be. More...
bool	**isAngularVelocityCompressed** = false Compress angular velocity floats when sending over the network. More...
bool	**isPositionCompressed** = false Compress position floats when sending over the network. More...
bool	**isRotationCompressed** = false Compress rotation floats when sending over the network. More...
bool	**isScaleCompressed** = false Compress scale floats when sending over the network. More...
bool	**isSmoothingAuthorityChanges** = false Smooths out authority changes. More...
bool	**isSyncingChild** = false Does this game object have a child object to sync? More...
bool	**isVelocityCompressed** = false Compress velocity floats when sending over the network. More...
Vector3	**lastAngularVelocityWhenStateWasSent** Angular velocity owner was at when the last angular velociy **State** was sent. More...
Vector3	**lastPositionWhenStateWasSent** Position owner was at when the last position **State** was sent. More...
Quaternion	**lastRotationWhenStateWasSent** = Quaternion.identity

	Rotation owner was at when the last rotation **State** was sent. More...
Vector3	**lastScaleWhenStateWasSent** Scale owner was at when the last scale **State** was sent. More...
float	**lastTimeStateWasSent** Last time owner sent a **State**. More...
Vector3	**lastVelocityWhenStateWasSent** Velocity owner was at when the last velocity **State** was sent. More...
Vector3	**latestReceivedAngularVelocity** The latest received angular velocity. Used for extrapolation. More...
Vector3	**latestReceivedVelocity** The latest received velocity. Used for extrapolation. More...
float	**maxPositionDifferenceForVelocitySyncing** = 10 An exponential scale used to determine how high the velocity should be set. More...
NetworkIdentity	**netID** Cached network ID. More...
int	**networkChannel** = Channels.DefaultUnreliable The channel to send network updates on. More...
int	**ownerChangeIndicator** = 1 Used to know when the owner has changed. Not an identifier. Only sent from Server. More...
float	**positionLerpSpeed** = .85f How fast to lerp the position to the target state. 0 is never, 1 is instant. More...
Rigidbody	**rb** Store a reference to the rigidbody so that we only have to call GetComponent() once. More...
Rigidbody2D	**rb2D** Store a reference to the 2D rigidbody so that we only have to call GetComponent() once. More...
GameObject	**realObjectToSync** Gets assigned to the real object to sync. Either this object or a child object. More...
float	**receivedPositionThreshold** = 0.0f The position won't be set on non-owned objects unless it changed this much. More...
float	**receivedRotationThreshold** = 0.0f The rotation won't be set on non-owned objects unless it changed this much. More...
int	**receivedStatesCounter** If this number is less than SendRate, force full time adjustment. Used when first entering a game. More...
float	**rotationLerpSpeed** = .85f

How fast to lerp the rotation to the target state. 0 is never, 1 is instant.. More...

float **scaleLerpSpeed** = .85f
How fast to lerp the scale to the target state. 0 is never, 1 is instant. More...

bool **sendAngularVelocity**
Variable we set at the beginning of Update so we only need to do the checks once a frame. More...

float **sendAngularVelocityThreshold** = 0.0f
The angular velocity won't send unless it changed this much. More...

bool **sendAtPositionalRestMessage** = false
Gets set to true when position is the same for two frames in order to tell non-owners to stop extrapolating position. More...

bool **sendAtRotationalRestMessage** = false
Gets set to true when rotation is the same for two frames in order to tell non-owners to stop extrapolating rotation. More...

bool **sendPosition**
Variable we set at the beginning of Update so we only need to do the checks once a frame. More...

float **sendPositionThreshold** = 0.0f
The position won't send unless it changed this much. More...

float **sendRate** = 30
How many times per second to send network updates. More...

bool **sendRotation**
Variable we set at the beginning of Update so we only need to do the checks once a frame. More...

float **sendRotationThreshold** = 0.0f
The rotation won't send unless it changed this much. More...

bool **sendScale**
Variable we set at the beginning of Update so we only need to do the checks once a frame. More...

float **sendScaleThreshold** = 0.0f
The scale won't send unless it changed this much. More...

bool **sendVelocity**
Variable we set at the beginning of Update so we only need to do the checks once a frame. More...

float **sendVelocityThreshold** = 0.0f
The velocity won't send unless it changed this much. More...

bool **setVelocityInsteadOfPositionOnNonOwners** = false
Set velocity on non-owners instead of the position. More...

float **snapPositionThreshold** = 0
If a synced object's position is more than snapPositionThreshold units from the target position, it will jump to the target position immediately instead of lerping. More...

float	**snapRotationThreshold** = 0 If a synced object's rotation is more than snapRotationThreshold from the target rotation, it will jump to the target rotation immediately instead of lerping. More...
float	**snapScaleThreshold** = 0 If a synced object's scale is more than snapScaleThreshold units from the target scale, it will jump to the target scale immediately instead of lerping. More...
float	**snapTimeThreshold** = 3.0f The estimated owner time of non-owned objects will change instantly if it is off by this amount. More...
State []	**stateBuffer** Non-owners keep a list of recent States received over the network for interpolating. More...
int	**stateCount** The number of States in the stateBuffer More...
SyncMode	**syncAngularVelocity** = SyncMode.XYZ Angular velocity sync mode More...
int	**syncIndex** = 0 Index to know which object to sync. More...
SyncMode	**syncPosition** = SyncMode.XYZ Position sync mode More...
SyncMode	**syncRotation** = SyncMode.XYZ Rotation sync mode More...
SyncMode	**syncScale** = SyncMode.XYZ Scale sync mode More...
SyncMode	**syncVelocity** = SyncMode.XYZ Velocity sync mode More...
float	**timeCorrectionSpeed** = .1f How fast to change the estimated owner time of non-owned objects. 0 is never, 5 is basically instant. More...
bool	**useExtrapolationDistanceLimit** = false Whether or not you want to use the extrapolationDistanceLimit. More...
bool	**useExtrapolationTimeLimit** = true Whether or not you want to use the extrapolationTimeLimit. More...
validateStateDelegate	**validateStateMethod** = **validateState** Validation method variable More...
WhenToUpdateTransform	**whenToUpdateTransform** = WhenToUpdateTransform.Update Where the object's Transform is updated on non-owners. More...

Properties

float	**approximateNetworkTimeOnOwner** `[get, set]` The current estimated time on the owner. More...

bool **isSyncingXAngularVelocity** `[get]`
Determine if should be syncing. More...

bool **isSyncingXPosition** `[get]`
Determine if should be syncing. More...

bool **isSyncingXRotation** `[get]`
Determine if should be syncing. More...

bool **isSyncingXScale** `[get]`
Determine if should be syncing. More...

bool **isSyncingXVelocity** `[get]`
Determine if should be syncing. More...

bool **isSyncingYAngularVelocity** `[get]`
Determine if should be syncing. More...

bool **isSyncingYPosition** `[get]`
Determine if should be syncing. More...

bool **isSyncingYRotation** `[get]`
Determine if should be syncing. More...

bool **isSyncingYScale** `[get]`
Determine if should be syncing. More...

bool **isSyncingYVelocity** `[get]`
Determine if should be syncing. More...

bool **isSyncingZAngularVelocity** `[get]`
Determine if should be syncing. More...

bool **isSyncingZPosition** `[get]`
Determine if should be syncing. More...

bool **isSyncingZRotation** `[get]`
Determine if should be syncing. More...

bool **isSyncingZScale** `[get]`
Determine if should be syncing. More...

bool **isSyncingZVelocity** `[get]`
Determine if should be syncing. More...

Detailed Description

Sync a Transform or Rigidbody over the network. Uses interpolation and extrapolation.

Overview: Owned objects send States. Owned objects use sendRate first and foremost to determine how often to send States. It will then defer to the thresholds to see if any of them have been passed and if so, it will send a **State** out to non-owners so that they have the updated Transform and Rigidbody information. Unowned objects receive States. Unowned objects will try to be interpolationBackTime (seconds) in the past and use the lerpSpeed variables to determine how fast to move from the current transform to the new transform. The new transform is determined by interpolating between received States. The object will start extrapolating if there are no new States to use (latency spike).

Member Enumeration Documentation

◆ ExtrapolationMode

enum **Smooth.SmoothSync.ExtrapolationMode** strong

Extrapolation type.

Extrapolation is going into the unknown based on information we had in the past. Generally, you'll want extrapolation to help fill in missing information during latency spikes. None - Use no extrapolation. Limited - Use the settings for extrapolation limits. Unlimited - Allow extrapolation forever. Must be syncing velocity in order to utilize extrapolation.

◆ WhenToUpdateTransform

enum **Smooth.SmoothSync.WhenToUpdateTransform** strong

Info to know where to update the Transform.

Member Function Documentation

◆ addState()

void Smooth.SmoothSync.addState (**State** state) inline

Add an incoming state to the stateBuffer on non-owned objects.

◆ Awake()

void Smooth.SmoothSync.Awake ( ) inline

Cache references to components.

◆ checkIfOwnerHasChanged()

void Smooth.SmoothSync.checkIfOwnerHasChanged ( ) inline

Checks if the owner has changed on each received **State**. If it has, add a "fake" received **State** to the **State** array with the current Transform so that you can lerp between it and the first **State** from the new owner.

◆ clearBuffer()

void Smooth.SmoothSync.clearBuffer ( ) inline

Clear the state buffer. Must be called on all non-owned objects if it's ownership has changed.

◆ CmdTeleport()

void Smooth.SmoothSync.CmdTeleport (Vector3 position,
Vector3 rotation,
Vector3 scale,
float tempOwnerTime
) inline

Echoes a teleport **State** from the host to all clients.

◆ forceStateSendNextFixedUpdate()

void Smooth.SmoothSync.forceStateSendNextFixedUpdate ( ) inline

Forces the **State** to be sent on owned objects the next time it goes through FixedUpdate().

The state will get sent next frame regardless of all limitations.

◆ getPosition()

Vector3 Smooth.SmoothSync.getPosition ( ) inline

Get position of object based on if child or not.

◆ getRotation()

Quaternion Smooth.SmoothSync.getRotation ( ) inline

Get rotation of object based on if child or not.

◆ getScale()

Vector3 Smooth.SmoothSync.getScale ( ) inline

Get scale of object.

◆ OnEnable()

void Smooth.SmoothSync.OnEnable ( ) inline

Automatically sends teleport message for this object **OnEnable()**.

◆ OnStartClient()

override void Smooth.SmoothSync.OnStartClient ( ) inline

Register network message handlers on clients.

◆ OnStartServer()

override void Smooth.SmoothSync.OnStartServer ( ) inline

Register network message handlers on server.

◆ RpcTeleport()

```
void Smooth.SmoothSync.RpcTeleport ( Vector3  position,
                                     Vector3  rotation,
                                     Vector3  scale,
                                     float    tempOwnerTime
                                   )
```

inline

Receive teleport **State** on clients and add to **State** array.

◆ setPosition()

```
void Smooth.SmoothSync.setPosition ( Vector3  position,
                                     bool     isTeleporting
                                   )
```

inline

Set position of object based on if child or not.

◆ setRotation()

```
void Smooth.SmoothSync.setRotation ( Quaternion  rotation,
                                     bool        isTeleporting
                                   )
```

inline

Set rotation of object based on if child or not.

◆ setScale()

```
void Smooth.SmoothSync.setScale ( Vector3  scale,
                                  bool     isTeleporting
                                )
```

inline

Set scale of object.

◆ shouldSendAngularVelocity()

bool Smooth.SmoothSync.shouldSendAngularVelocity ( ) inline

Check if angular velocity has changed enough.

If sendAngularVelocityThreshold is 0, returns true if the current angular velocity is different from the latest sent angular velocity. If sendAngularVelocityThreshold is greater than 0, returns true if difference between angular velocity and latest sent angular velocity is greater than the angular velocity threshold.

◆ shouldSendPosition()

bool Smooth.SmoothSync.shouldSendPosition ( ) inline

Check if position has changed enough.

If sendPositionThreshold is 0, returns true if the current position is different than the latest sent position. If sendPositionThreshold is greater than 0, returns true if distance between position and latest sent position is greater than the sendPositionThreshold.

◆ shouldSendRotation()

bool Smooth.SmoothSync.shouldSendRotation ( ) inline

Check if rotation has changed enough.

If sendRotationThreshold is 0, returns true if the current rotation is different from the latest sent rotation. If sendRotationThreshold is greater than 0, returns true if difference (angle) between rotation and latest sent rotation is greater than the sendRotationThreshold.

◆ shouldSendScale()

bool Smooth.SmoothSync.shouldSendScale ( ) inline

Check if scale has changed enough.

If sendScaleThreshold is 0, returns true if the current scale is different than the latest sent scale. If sendScaleThreshold is greater than 0, returns true if the difference between scale and latest sent scale is greater than the sendScaleThreshold.

◆ shouldSendVelocity()

bool Smooth.SmoothSync.shouldSendVelocity ( ) inline

Check if velocity has changed enough.

If sendVelocityThreshold is 0, returns true if the current velocity is different from the latest sent velocity. If sendVelocityThreshold is greater than 0, returns true if difference between velocity and latest sent velocity is greater than the velocity threshold.

◆ stopLerping()

void Smooth.SmoothSync.stopLerping ( ) inline

Stop updating the States of non-owned objects so that the object can be teleported.

◆ teleport()

void Smooth.SmoothSync.teleport ( ) inline

Deprecated. Use **teleportOwnedObjectFromOwner()** or **teleportAnyObjectFromServer()**.

◆ teleportAnyObjectFromServer()

void Smooth.SmoothSync.teleportAnyObjectFromServer (Vector3 newPosition,
Quaternion newRotation,
Vector3 newScale
) inline

Teleport the object, the transform will not be interpolated on non-owners.

Call **teleportAnyObjectFromServer()** on any object to teleport that object on all systems. Full example of use in the example scene in SmoothSyncExamplePlayerController.cs.

◆ teleportOwnedObjectFromOwner()

void Smooth.SmoothSync.teleportOwnedObjectFromOwner ( ) inline

Teleport the object, the transform will not be interpolated on non-owners.

Call **teleportOwnedObjectFromOwner()** on any owned object to send it's current transform to non-owners, telling them to teleport. Full example of use in the example scene in SmoothSyncExamplePlayerController.cs.

◆ validateState()

static bool Smooth.SmoothSync.validateState (**State** latestReceivedState,
State latestValidatedState
) inline static

Validation method

The default validation method that allows all States. Your custom validation method must match the parameter types of this method. Return false to deny the **State**. The **State** will not be added locally on the server and it will not be sent out to other clients. Return true to accept the **State**. The **State** will be added locally on the server and will be sent out to other clients.

◆ validateStateDelegate()

delegate bool Smooth.SmoothSync.validateStateDelegate (**State** receivedState,
State latestVerifiedState
)

Validation delegate

To tie in your own validation method, check the SmoothSyncExample scene and SmoothSyncExamplePlayerController.cs on how to use the validation delegate.

Smooth Sync will call this on the server on every incoming **State** message. By default it allows every received **State** but you can set the validateStateMethod to a custom one in order to validate that the clients aren't modifying their owned objects beyond the game's intended limits.

Member Data Documentation

◆ childObjectSmoothSyncs

SmoothSync [] Smooth.SmoothSync.childObjectSmoothSyncs = new **SmoothSync**[0]

Reference to child objects so you can compare to syncIndex.

◆ childObjectToSync

GameObject Smooth.SmoothSync.childObjectToSync

Child object to sync

Leave blank if you want to sync this object. In order to sync a child object, you must add two instances of **SmoothSync** to the parent. Set childObjectToSync on one of them to point to the child you want to sync and leave it blank on the other to sync the parent. You cannot sync children without syncing the parent.

◆ extrapolationDistanceLimit

float Smooth.SmoothSync.extrapolationDistanceLimit = 20.0f

How much distance into the future a non-owned object is allowed to extrapolate.

Extrapolating too far tends to cause erratic and non-realistic movement, but a little bit of extrapolation is better than none because it keeps things working semi-right during latency spikes.

Must be syncing velocity in order to utilize extrapolation.

Measured in distance units.

◆ extrapolationMode

ExtrapolationMode Smooth.SmoothSync.extrapolationMode = ExtrapolationMode.Limited

The amount of extrapolation used.

Extrapolation is going into the unknown based on information we had in the past. Generally, you'll want extrapolation to help fill in missing information during lag spikes. None - Use no extrapolation. Limited - Use the settings for extrapolation limits. Unlimited - Allow extrapolation forever.

◆ extrapolationTimeLimit

float Smooth.SmoothSync.extrapolationTimeLimit = 5.0f

How much time into the future a non-owned object is allowed to extrapolate.

Extrapolating too far tends to cause erratic and non-realistic movement, but a little bit of extrapolation is better than none because it keeps things working semi-right during latency spikes.

Must be syncing velocity in order to utilize extrapolation.

Measured in seconds.

◆ forceStateSend

bool Smooth.SmoothSync.forceStateSend = false

Gets set to true in order to force the **State** to be sent next frame on owners.

◆ hasRigidbody

bool Smooth.SmoothSync.hasRigidbody = false

Does this game object have a Rigidbody component?

Is much less resource intensive to check a boolean than if a component exists.

◆ hasRigidbody2D

bool Smooth.SmoothSync.hasRigidbody2D = false

Does this game object have a Rigidbody2D component?

Is much less resource intensive to check a boolean than if a component exists.

◆ interpolationBackTime

float Smooth.SmoothSync.interpolationBackTime = .1f

How much time in the past non-owned objects should be.

interpolationBackTime is the amount of time in the past the object will be on non-owners. This is so if you hit a latency spike, you still have a buffer of the interpolation back time of known States before you start extrapolating into the unknown.

Essentially, for everyone who has ping less than the interpolationBackTime, the object will appear in the same place on all screens.

Increasing this will make interpolation more likely to be used, which means the synced position will be more likely to be an actual position that the owner was at.

Decreasing this will make extrapolation more likely to be used, this will increase responsiveness, but with any latency spikes that last longer than the interpolationBackTime, the position will be less correct to where the player was actually at.

Keep above 1/SendRate to attempt to always interpolate.

Measured in seconds.

◆ isAngularVelocityCompressed

bool Smooth.SmoothSync.isAngularVelocityCompressed = false

Compress angular velocity floats when sending over the network.

Convert angular velocity floats sent over the network to Halfs, which use half as much bandwidth but are also half as precise.

◆ isPositionCompressed

bool Smooth.SmoothSync.isPositionCompressed = false

Compress position floats when sending over the network.

Convert position floats sent over the network to Halfs, which use half as much bandwidth but are also half as precise. It'll start becoming noticeably "off" over ~600.

◆ isRotationCompressed

bool Smooth.SmoothSync.isRotationCompressed = false

Compress rotation floats when sending over the network.

Convert rotation floats sent over the network to Halfs, which use half as much bandwidth but are also half as precise.

◆ isScaleCompressed

bool Smooth.SmoothSync.isScaleCompressed = false

Compress scale floats when sending over the network.

Convert scale floats sent over the network to Halfs, which use half as much bandwidth but are also half as precise.

◆ isSmoothingAuthorityChanges

bool Smooth.SmoothSync.isSmoothingAuthorityChanges = false

Smooths out authority changes.

Sends an extra byte with owner information so we can know when the owner has changed and smooth accordingly.

◆ isSyncingChild

bool Smooth.SmoothSync.isSyncingChild = false

Does this game object have a child object to sync?

Is much less resource intensive to check a boolean than if a Gameobject exists.

◆ isVelocityCompressed

bool Smooth.SmoothSync.isVelocityCompressed = false

Compress velocity floats when sending over the network.

Convert velocity floats sent over the network to Halfs, which use half as much bandwidth but are also half as precise.

◆ lastAngularVelocityWhenStateWasSent

Vector3 Smooth.SmoothSync.lastAngularVelocityWhenStateWasSent

Angular velocity owner was at when the last angular velociy **State** was sent.

◆ lastPositionWhenStateWasSent

Vector3 Smooth.SmoothSync.lastPositionWhenStateWasSent

Position owner was at when the last position **State** was sent.

◆ lastRotationWhenStateWasSent

Quaternion Smooth.SmoothSync.lastRotationWhenStateWasSent = Quaternion.identity

Rotation owner was at when the last rotation **State** was sent.

◆ lastScaleWhenStateWasSent

Vector3 Smooth.SmoothSync.lastScaleWhenStateWasSent

Scale owner was at when the last scale **State** was sent.

◆ lastTimeStateWasSent

float Smooth.SmoothSync.lastTimeStateWasSent

Last time owner sent a **State**.

◆ lastVelocityWhenStateWasSent

Vector3 Smooth.SmoothSync.lastVelocityWhenStateWasSent

Velocity owner was at when the last velocity **State** was sent.

◆ latestReceivedAngularVelocity

Vector3 Smooth.SmoothSync.latestReceivedAngularVelocity

The latest received angular velocity. Used for extrapolation.

◆ latestReceivedVelocity

Vector3 Smooth.SmoothSync.latestReceivedVelocity

The latest received velocity. Used for extrapolation.

◆ maxPositionDifferenceForVelocitySyncing

float Smooth.SmoothSync.maxPositionDifferenceForVelocitySyncing = 10

An exponential scale used to determine how high the velocity should be set.

If the difference between where it should be and where it is hits this, then it will automatically jump to location. Is on an exponential scale normally.

◆ netID

NetworkIdentity Smooth.SmoothSync.netID

Cached network ID.

◆ networkChannel

int Smooth.SmoothSync.networkChannel = Channels.DefaultUnreliable

The channel to send network updates on.

◆ ownerChangeIndicator

int Smooth.SmoothSync.ownerChangeIndicator = 1

Used to know when the owner has changed. Not an identifier. Only sent from Server.

◆ positionLerpSpeed

float Smooth.SmoothSync.positionLerpSpeed = .85f

How fast to lerp the position to the target state. 0 is never, 1 is instant.

Lower values mean smoother but maybe sluggish movement. Higher values mean more responsive but maybe jerky or stuttery movement.

◆ rb

Rigidbody Smooth.SmoothSync.rb

Store a reference to the rigidbody so that we only have to call GetComponent() once.

Will automatically use Rigidbody or Rigidbody2D depending on what is on the game object.

◆ rb2D

Rigidbody2D Smooth.SmoothSync.rb2D

Store a reference to the 2D rigidbody so that we only have to call GetComponent() once.

◆ realObjectToSync

GameObject Smooth.SmoothSync.realObjectToSync

Gets assigned to the real object to sync. Either this object or a child object.

◆ receivedPositionThreshold

float Smooth.SmoothSync.receivedPositionThreshold = 0.0f

The position won't be set on non-owned objects unless it changed this much.

Set to 0 to always update the position of non-owned objects if it has changed, and to use one less Vector3.Distance() check per frame if you also have snapPositionThreshold at 0. If greater than 0, a synced object's position is only updated if it is off from the target position by more than the threshold. Usually keep this at 0 or really low, at higher numbers it's useful if you are extrapolating into the future and want to stop instantly and not have it backtrack to where it currently is on the owner. Measured in distance units.

◆ receivedRotationThreshold

float Smooth.SmoothSync.receivedRotationThreshold = 0.0f

The rotation won't be set on non-owned objects unless it changed this much.

Set to 0 to always update the rotation of non-owned objects if it has changed, and to use one less Quaternion.Angle() check per frame if you also have snapRotationThreshold at 0. If greater than 0, a synced object's rotation is only updated if it is off from the target rotation by more than the threshold. Usually keep this at 0 or really low, at higher numbers it's useful if you are extrapolating into the future and want to stop instantly and not have it backtrack to where it currently is on the owner. Measured in degrees.

◆ receivedStatesCounter

int Smooth.SmoothSync.receivedStatesCounter

If this number is less than SendRate, force full time adjustment. Used when first entering a game.

◆ rotationLerpSpeed

float Smooth.SmoothSync.rotationLerpSpeed = .85f

How fast to lerp the rotation to the target state. 0 is never, 1 is instant..

Lower values mean smoother but maybe sluggish movement. Higher values mean more responsive but maybe jerky or stuttery movement.

◆ scaleLerpSpeed

float Smooth.SmoothSync.scaleLerpSpeed = .85f

How fast to lerp the scale to the target state. 0 is never, 1 is instant.

Lower values mean smoother but maybe sluggish movement. Higher values mean more responsive but maybe jerky or stuttery movement.

◆ sendAngularVelocity

bool Smooth.SmoothSync.sendAngularVelocity

Variable we set at the beginning of Update so we only need to do the checks once a frame.

◆ sendAngularVelocityThreshold

float Smooth.SmoothSync.sendAngularVelocityThreshold = 0.0f

The angular velocity won't send unless it changed this much.

Set to 0 to always send the angular velocity of owned objects if it has changed since the last sent angular velocity, and for a hardware performance increase, but at the cost of network usage. If greater than 0, a synced object's angular velocity is only sent if its angular velocity is off from the last sent angular velocity by more than the threshold. Measured in degrees per second.

◆ sendAtPositionalRestMessage

bool Smooth.SmoothSync.sendAtPositionalRestMessage = false

Gets set to true when position is the same for two frames in order to tell non-owners to stop extrapolating position.

◆ sendAtRotationalRestMessage

bool Smooth.SmoothSync.sendAtRotationalRestMessage = false

Gets set to true when rotation is the same for two frames in order to tell non-owners to stop extrapolating rotation.

◆ sendPosition

bool Smooth.SmoothSync.sendPosition

Variable we set at the beginning of Update so we only need to do the checks once a frame.

◆ sendPositionThreshold

float Smooth.SmoothSync.sendPositionThreshold = 0.0f

The position won't send unless it changed this much.

Set to 0 to always send the position of owned objects if it has changed since the last sent position, and for a hardware performance increase, but at the cost of network usage. If greater than 0, a synced object's position is only sent if its position is off from the last sent position by more than the threshold. Measured in distance units.

◆ sendRate

float Smooth.SmoothSync.sendRate = 30

How many times per second to send network updates.

For low send rates, try lowering the lerpSpeeds if it is too jittery. Keeping your interpolationBackTime larger than your send rate interval will be good for interpolation.

◆ sendRotation

bool Smooth.SmoothSync.sendRotation

Variable we set at the beginning of Update so we only need to do the checks once a frame.

◆ sendRotationThreshold

float Smooth.SmoothSync.sendRotationThreshold = 0.0f

The rotation won't send unless it changed this much.

Set to 0 to always send the rotation of owned objects if it has changed since the last sent rotation, and for a hardware performance increase, but at the cost of network usage. If greater than 0, a synced object's rotation is only sent if its rotation is off from the last sent rotation by more than the threshold. Measured in degrees.

◆ sendScale

bool Smooth.SmoothSync.sendScale

Variable we set at the beginning of Update so we only need to do the checks once a frame.

◆ sendScaleThreshold

float Smooth.SmoothSync.sendScaleThreshold = 0.0f

The scale won't send unless it changed this much.

Set to 0 to always send the scale of owned objects if it has changed since the last sent scale, and for a hardware performance increase, but at the cost of network usage. If greater than 0, a synced object's scale is only sent if its scale is off from the last sent scale by more than the threshold. Measured in distance units.

◆ sendVelocity

bool Smooth.SmoothSync.sendVelocity

Variable we set at the beginning of Update so we only need to do the checks once a frame.

◆ sendVelocityThreshold

float Smooth.SmoothSync.sendVelocityThreshold = 0.0f

The velocity won't send unless it changed this much.

Set to 0 to always send the velocity of owned objects if it has changed since the last sent velocity, and for a hardware performance increase, but at the cost of network usage. If greater than 0, a synced object's velocity is only sent if its velocity is off from the last sent velocity by more than the threshold. Measured in velocity units.

◆ setVelocityInsteadOfPositionOnNonOwners

bool Smooth.SmoothSync.setVelocityInsteadOfPositionOnNonOwners = false

Set velocity on non-owners instead of the position.

Requires Rigidbody. Uses the synced position to determine what to set the velocity to on unowned objects. This will produce smoother results at faster speeds and was made for games like flying or racing. Is less accurate than default **Smooth** Sync. Things can also go wrong if the position is blocked that it is trying to get to. You should use a "Snap Position Threshold" if you use this.

◆ snapPositionThreshold

float Smooth.SmoothSync.snapPositionThreshold = 0

If a synced object's position is more than snapPositionThreshold units from the target position, it will jump to the target position immediately instead of lerping.

Set to zero to not use at all and use one less Vector3.Distance() check per frame if you also have receivedPositionThreshold at 0. Measured in distance units.

◆ snapRotationThreshold

float Smooth.SmoothSync.snapRotationThreshold = 0

If a synced object's rotation is more than snapRotationThreshold from the target rotation, it will jump to the target rotation immediately instead of lerping.

Set to zero to not use at all and use one less Quaternion.Angle() check per frame if you also have receivedRotationThreshold at 0. Measured in degrees.

◆ snapScaleThreshold

float Smooth.SmoothSync.snapScaleThreshold = 0

If a synced object's scale is more than snapScaleThreshold units from the target scale, it will jump to the target scale immediately instead of lerping.

Set to zero to not use at all and use one less Vector3.Distance() check per frame. Measured in distance units.

◆ snapTimeThreshold

float Smooth.SmoothSync.snapTimeThreshold = 3.0f

The estimated owner time of non-owned objects will change instantly if it is off by this amount.

The estimated owner time will change instantly if the difference is larger than this amount (in seconds) when receiving an update. Generally keep at default unless you have a very low send rate and expect large variance in your latencies.

◆ stateBuffer

State [] Smooth.SmoothSync.stateBuffer

Non-owners keep a list of recent States received over the network for interpolating.

Index 0 is the newest received **State**.

◆ stateCount

int Smooth.SmoothSync.stateCount

The number of States in the stateBuffer

◆ syncAngularVelocity

SyncMode Smooth.SmoothSync.syncAngularVelocity = SyncMode.XYZ

Angular velocity sync mode

Fine tune how angular velocity is synced.

◆ syncIndex

int Smooth.SmoothSync.syncIndex = 0

Index to know which object to sync.

◆ syncPosition

SyncMode Smooth.SmoothSync.syncPosition = SyncMode.XYZ

Position sync mode

Fine tune how position is synced. For objects that don't move, use SyncMode.NONE

◆ syncRotation

SyncMode Smooth.SmoothSync.syncRotation = SyncMode.XYZ

Rotation sync mode

Fine tune how rotation is synced. For objects that don't rotate, use SyncMode.NONE

◆ syncScale

SyncMode Smooth.SmoothSync.syncScale = SyncMode.XYZ

Scale sync mode

Fine tune how scale is synced. For objects that don't scale, use SyncMode.NONE

◆ syncVelocity

SyncMode Smooth.SmoothSync.syncVelocity = SyncMode.XYZ

Velocity sync mode

Fine tune how velocity is synced.

◆ timeCorrectionSpeed

float Smooth.SmoothSync.timeCorrectionSpeed = .1f

How fast to change the estimated owner time of non-owned objects. 0 is never, 5 is basically instant.

The estimated owner time can shift by this amount per second. Lower values will be smoother but it may take longer to adjust to larger jumps in latency. Probably keep this lower than ~.5 unless you are having serious latency variance issues.

◆ useExtrapolationDistanceLimit

bool Smooth.SmoothSync.useExtrapolationDistanceLimit = false

Whether or not you want to use the extrapolationDistanceLimit.

You can use only the extrapolationTimeLimit and save a distance check every extrapolation frame. Must be syncing velocity in order to utilize extrapolation.

◆ useExtrapolationTimeLimit

bool Smooth.SmoothSync.useExtrapolationTimeLimit = true

Whether or not you want to use the extrapolationTimeLimit.

You can use only the extrapolationTimeLimit and save a distance check every extrapolation frame. Must be syncing velocity in order to utilize extrapolation.

◆ validateStateMethod

validateStateDelegate Smooth.SmoothSync.validateStateMethod = **validateState**

Validation method variable

Holds a reference to the method that will be called to validate incoming States. You will set it to your custom validation method. It will be something like smoothSync.validateStateMethod = myCoolCustomValidatePlayerMethod; in the Start or Awake method of your object's script.

◆ whenToUpdateTransform

WhenToUpdateTransform Smooth.SmoothSync.whenToUpdateTransform = WhenToUpdateTransform.Update

Where the object's Transform is updated on non-owners.

Update will have smoother results but FixedUpdate might be better for physics.

Property Documentation

◆ approximateNetworkTimeOnOwner

float Smooth.SmoothSync.approximateNetworkTimeOnOwner [get] [set]

The current estimated time on the owner.

Time comes from the owner in every sync message. When it is received we set _ownerTime and lastTimeOwnerTimeWasSet. Then when we want to know what time it is we add time elapsed to the last _ownerTime we received.

◆ isSyncingXAngularVelocity

bool Smooth.SmoothSync.isSyncingXAngularVelocity [get]

Determine if should be syncing.

◆ isSyncingXPosition

bool Smooth.SmoothSync.isSyncingXPosition [get]

Determine if should be syncing.

◆ isSyncingXRotation

bool Smooth.SmoothSync.isSyncingXRotation [get]

Determine if should be syncing.

◆ isSyncingXScale

bool Smooth.SmoothSync.isSyncingXScale [get]

Determine if should be syncing.

◆ isSyncingXVelocity

bool Smooth.SmoothSync.isSyncingXVelocity get

Determine if should be syncing.

◆ isSyncingYAngularVelocity

bool Smooth.SmoothSync.isSyncingYAngularVelocity get

Determine if should be syncing.

◆ isSyncingYPosition

bool Smooth.SmoothSync.isSyncingYPosition get

Determine if should be syncing.

◆ isSyncingYRotation

bool Smooth.SmoothSync.isSyncingYRotation get

Determine if should be syncing.

◆ isSyncingYScale

bool Smooth.SmoothSync.isSyncingYScale get

Determine if should be syncing.

◆ isSyncingYVelocity

bool Smooth.SmoothSync.isSyncingYVelocity get

Determine if should be syncing.

◆ isSyncingZAngularVelocity

bool Smooth.SmoothSync.isSyncingZAngularVelocity get

Determine if should be syncing.

◆ isSyncingZPosition

bool Smooth.SmoothSync.isSyncingZPosition get

Determine if should be syncing.

◆ isSyncingZRotation

bool Smooth.SmoothSync.isSyncingZRotation get

Determine if should be syncing.

◆ isSyncingZScale

bool Smooth.SmoothSync.isSyncingZScale get

Determine if should be syncing.

◆ isSyncingZVelocity

bool Smooth.SmoothSync.isSyncingZVelocity get

Determine if should be syncing.