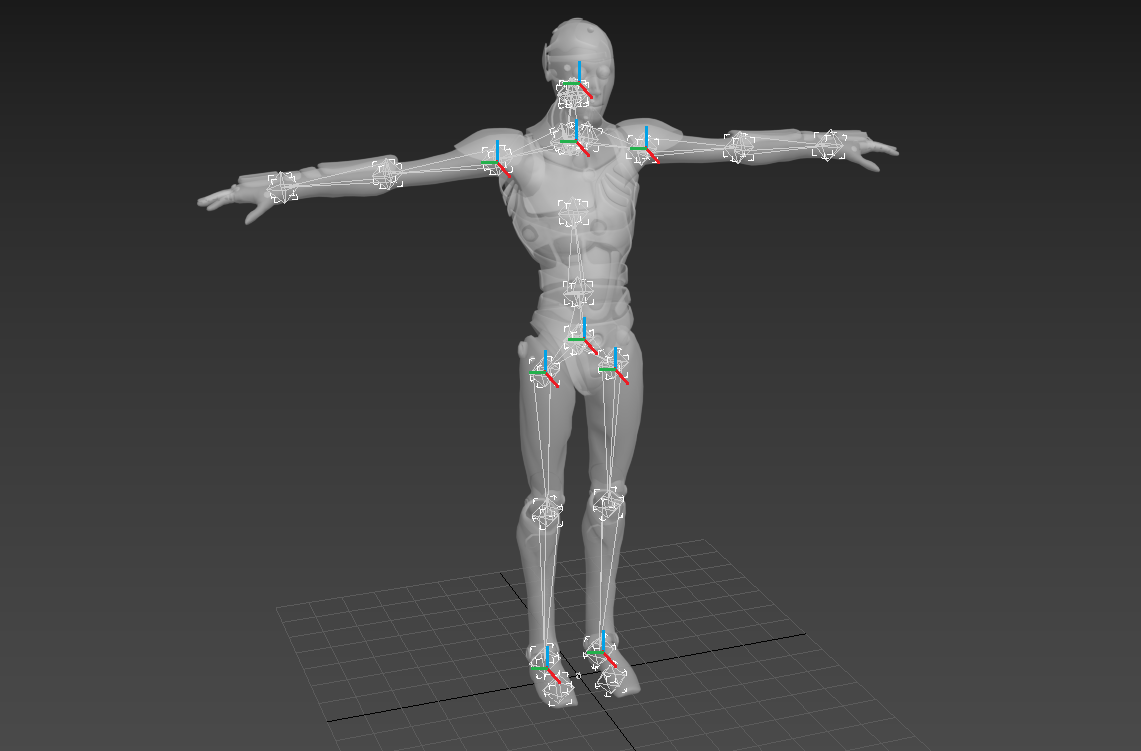
VR IK Body

# How to set up VR project to use VR IK Body

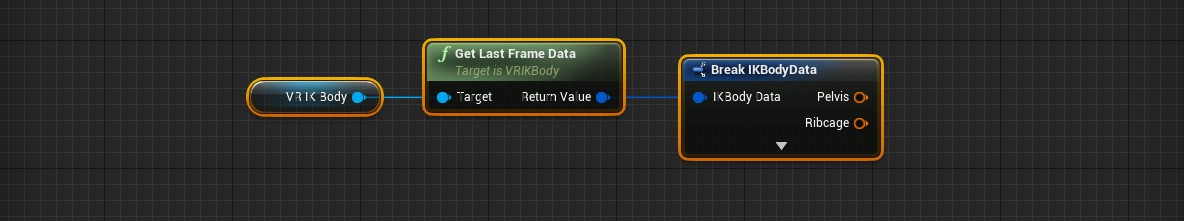
Import your skeletal mesh and create an animation blueprint.

Open you player pawn blueprint. Add skeletal mesh component, set its **Skeletal Mesh** (your skeletal mesh), **Animation Mode** (Use Animation Blueprint) and **Anim Class** (your animation blueprint) properties.

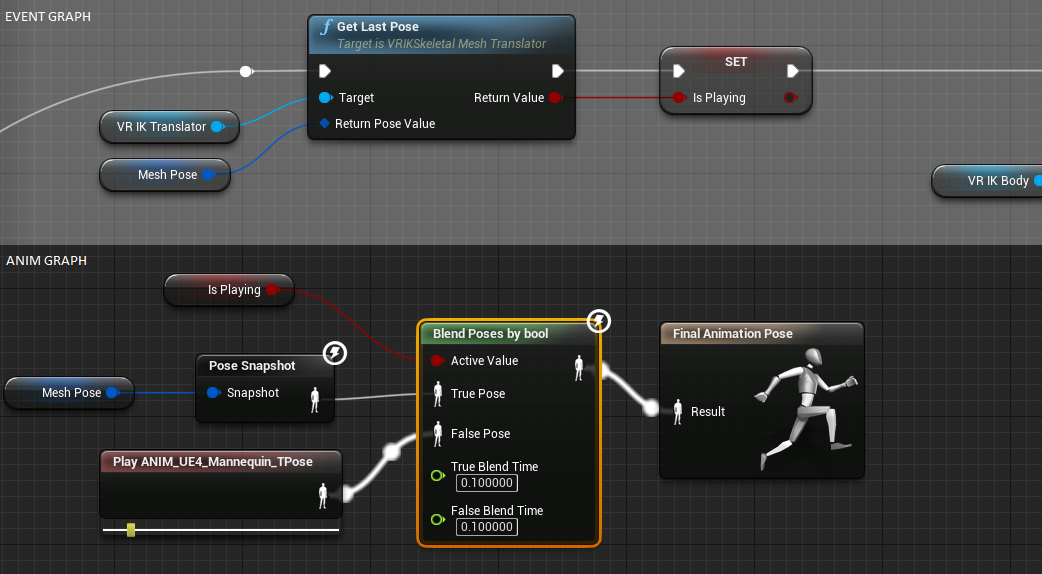
Add two actor components: VRIKBody and VRIKSkeletalMeshTranslator. VRIKBody is responsible for body calibration and calculating instant body state adjusted to player’s size. It returns transforms for some virtual bones in world space and for the follow bone rotators:



You can get access to this data by calling VRIKBody::GetLastFrameData



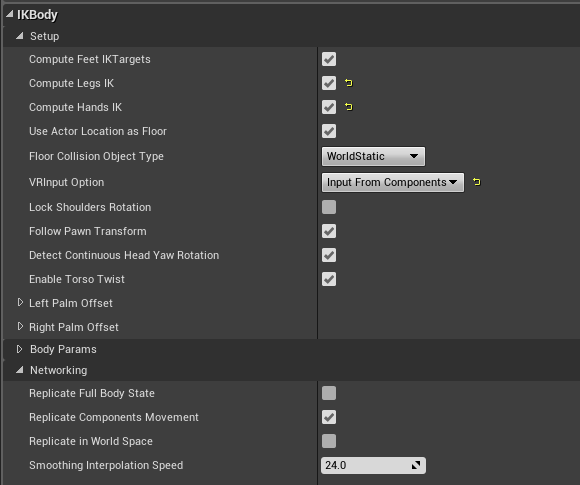
VRIKSkeletalMeshTranslator translates this virtual skeleton to any custom skeletal mesh. It returns PoseSnapshot object.



Simple animation blueprint setup is presented above. **VR IK Translator** is a reference to player’s VRIKSkeletalMeshTranslator component. Mesh Pose is a **PoseSnapshot** variable.

Both VRIKBody and VRIKSkeletalMeshTranslator components must be initialized in player pawn’s blueprint BeginPlay event.

## Major VRIKBody properties



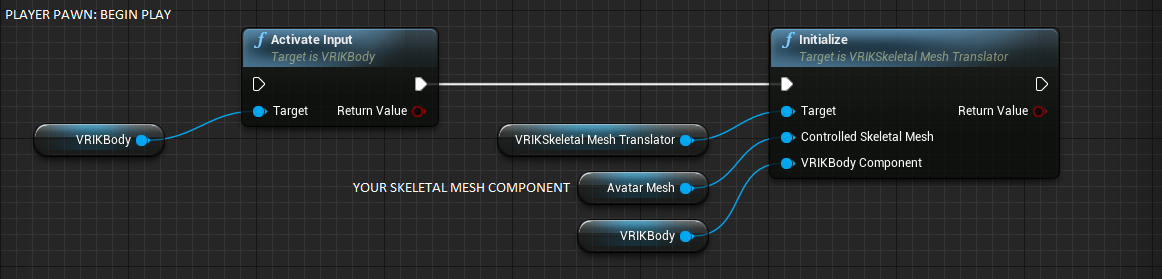
Check **ComputeLegsIK** and **ComputeHandIK** to enable IK calculations.

There are two ground detection methods.

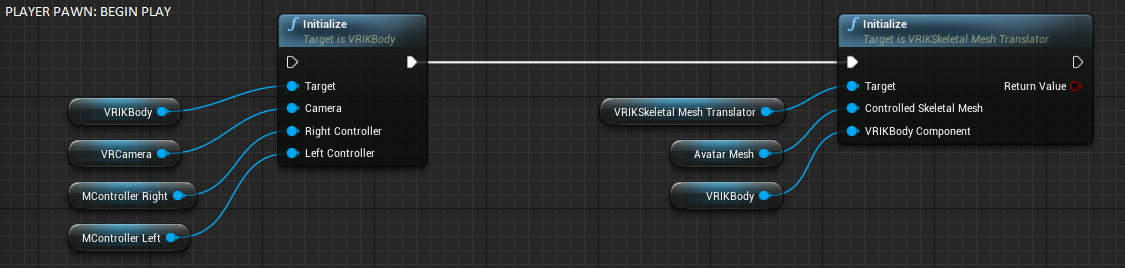
* **UseActorLocationAsFloor** is true. VRIKBody assume GetActorLocation().Z is a ground level.
* **UseActorLocationAsFloor** is false. VRIKBody perform LineTraceByObjectType to detect ground level below head. **FloorCollisionObjectType** must be set.

There are three **VR input Option** settings.

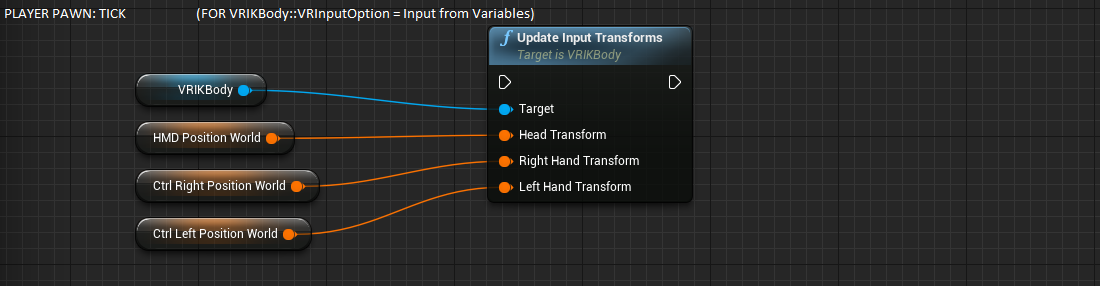
* Direct VR Input. The components takes HMD and motion controllers positions from UE4 VR API. VRIKBody component must be initialized by ActivateInput() call.



* Input from components. This option is useful if you already have camera and motion controller components and/or want them smoothly replicated in networking. You also can use simple scene and primitive components with this option to emulate VR input. Use Initialize(…) call instead of ActivateInput() to initialize references to input components in the VRIKBody.

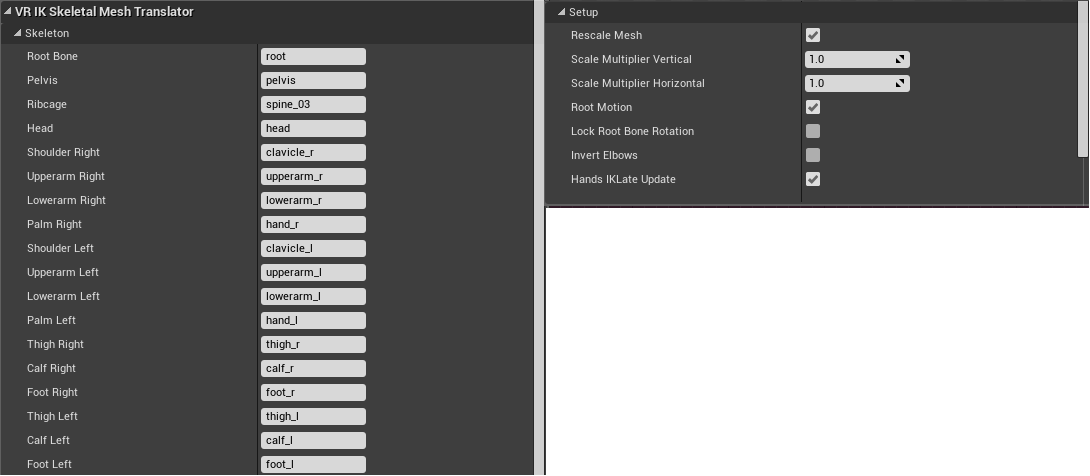


* Input from variables. Another way to simulate VR input. Initialize as Direct VR Input and call UpdateInputTransforms in Tick. This method doesn’t work with replication.



**LeftPalmOffset** and **RightPalmOffset** variables set constant hand position relative to motion controllers. Default values are for Vive wands controlles. Edit this values Oculus Touch controlles.

## Major VRIKSkeletalMeshTranslator properties



*Skeleton* subcategory contains the map of skeleton bone names. If you skeletal mesh doesn’t have separate root bone, set it to None (empty).

Uncheck **RescaleMesh** to reset skeletal mesh scale to 1 unit.

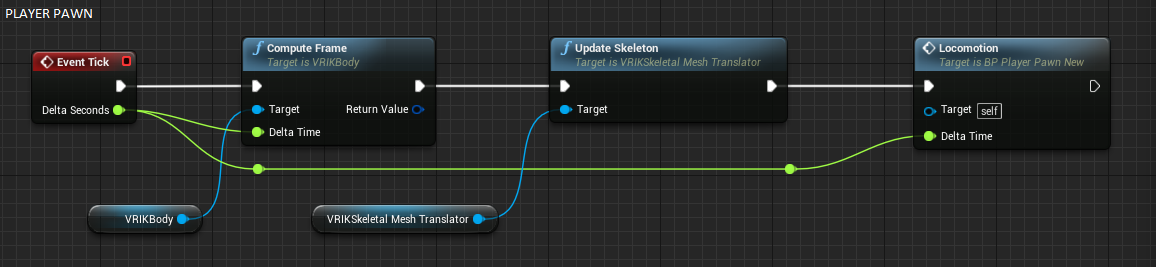
If **RescaleMesh** is true, you can apply additional adjustment to skeletal mesh scale by **ScaleMultiplierVertical** and **ScaleMultiplierHorizontal** variables.

**RootMotion** = true forces VRIKSkeletalMeshTranslator to update location of rotation of the controlled skeletal mesh component.

**Use InvertElbows** if elbows are bended in a wrong direction.

**HandsIKLateUpdate** forces VRIKSkeletalMeshTranslator to apply additional hands IK calculation when building PoseSnapshot to make sure hands are in an accurate positions. Uncheck it to save CPU time if you manually update hands by TwoBoneIK nodes in your animation blueprint.

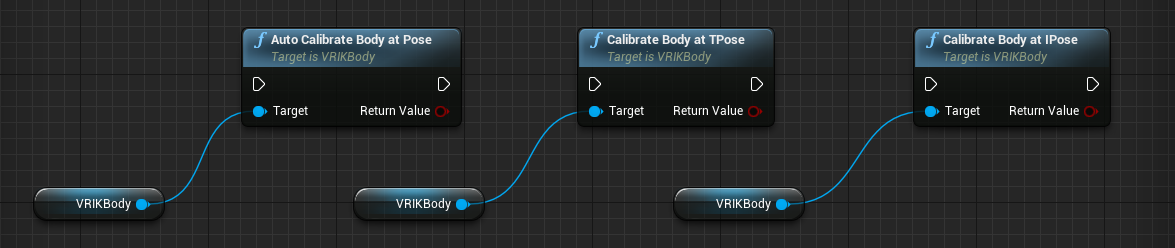
## Body calculation

To calculate current body state, call VRIKBody::ComputeFrame and VRIKSkeletalMeshTranslator::UpdateSkeleton.

Note. If you use sliding locomotion from trackpad input, keep in mind that functions like AddActorWorldOffset or AddActorRelativeOffset or CharacterMovementComponent change actor position after animation apply. It cause delay in skeletal mesh location update. SetActorLocation(…), at the other hand, updates location instantly, so it’s recommended.

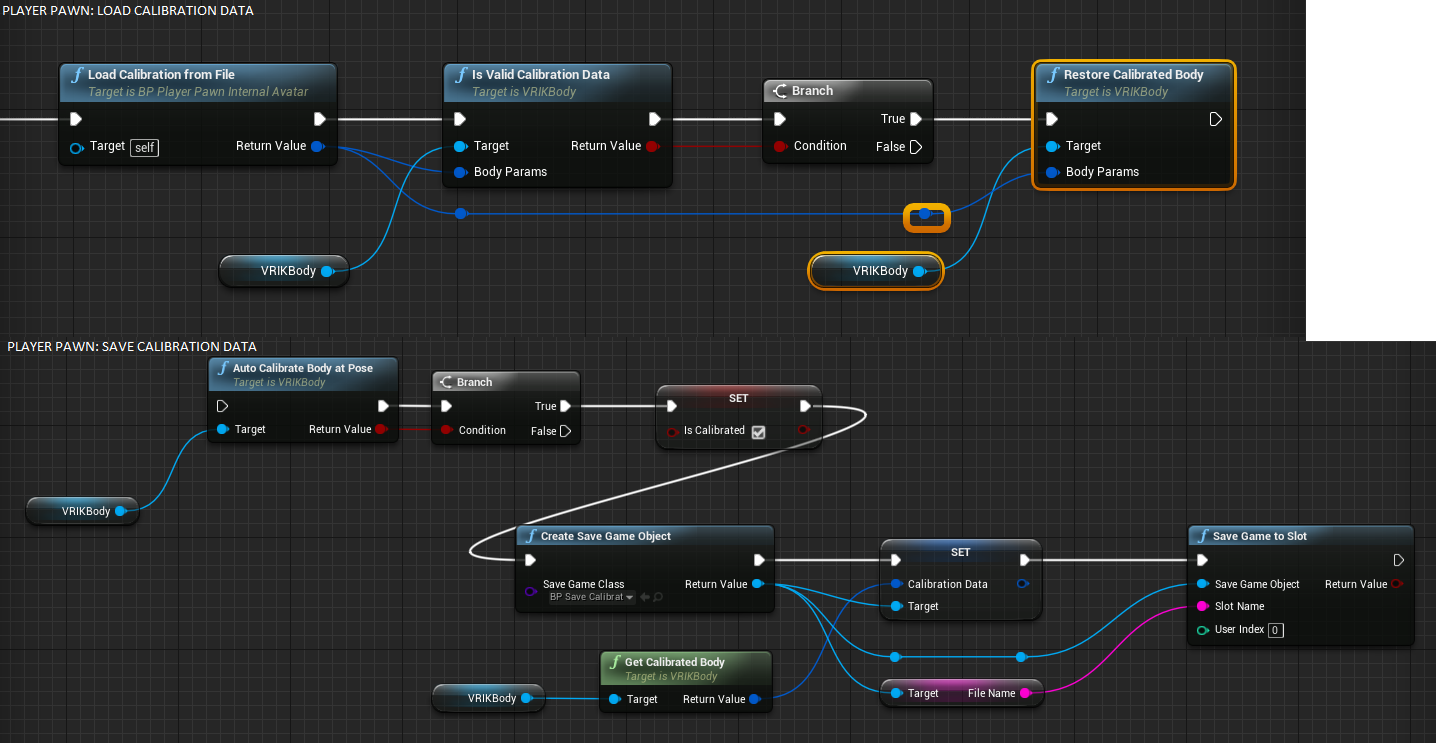
## Calibration

The plugin requires two-step calibration: hands to the left and right (T-Pose) and hand down (I pose). To perform calibration, call VRIKBody::AutoCalibrateBodyAtPose or VRIKBody::CalibrateBodyAtTPose or VRIKBody::CalibrateBodyAtIPose. All this functions return true if calibration is fully complete for both poses.



Note that VRIKSkeletalMeshTranslator requires body calibration. Call VRIKSkeletalMeshTranslator::ForceSetComponentAsCalibrated to activate it with default body params.

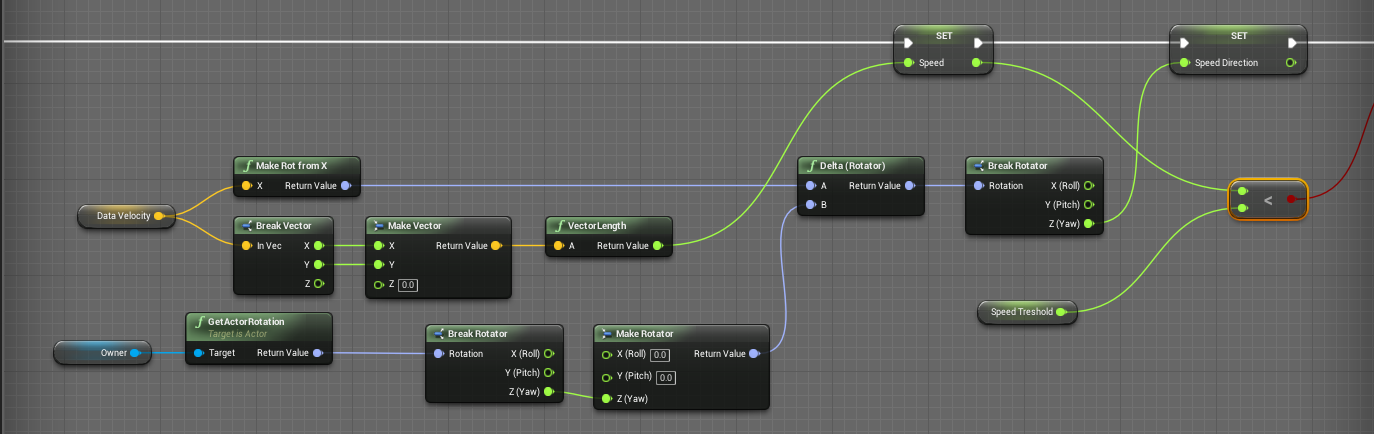
You can get current body calibration data by calling VRIKBody::GetCalibratedBody and load it by VRIKBody::RestoreCalibratedBody. Use this functions to save and load calibration data between sessions. IsValidCalibrationData is useful to check if loaded BodyParams value is valid.



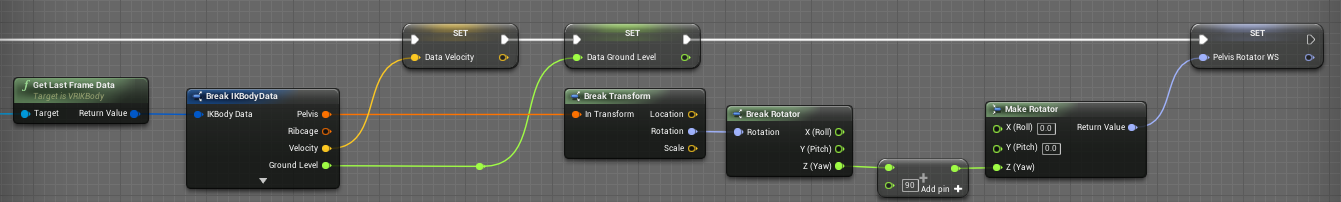
## Using strafe animation for legs

The plugin provides some tools to use custom legs animation instead of default walking cycle. Follow this steps to achieve it:

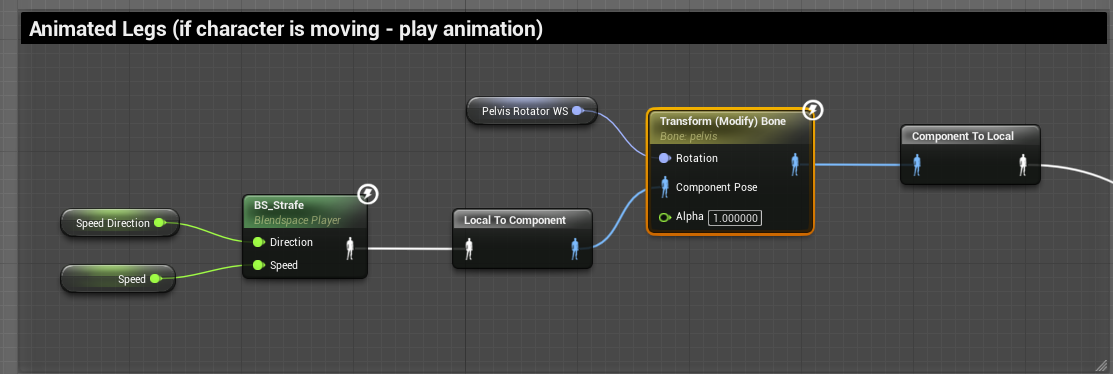
* Use VRIKBody::GetLastFrameData to get current velocity (*DataVelocity*) vector and calculate walking direction (Yaw) relative to torso forward direction.



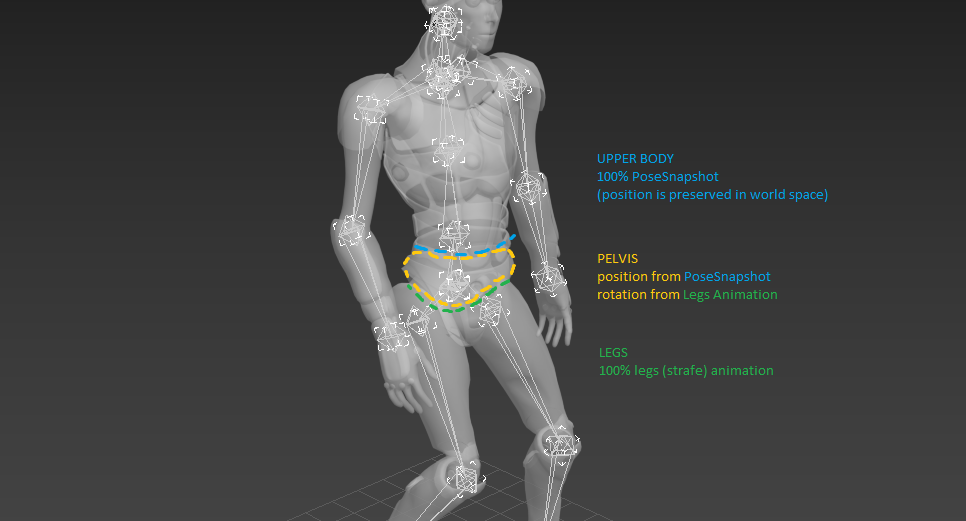
* Use VRIKBody::GetLastFrameData to calculate current torso direction (Yaw) in world space

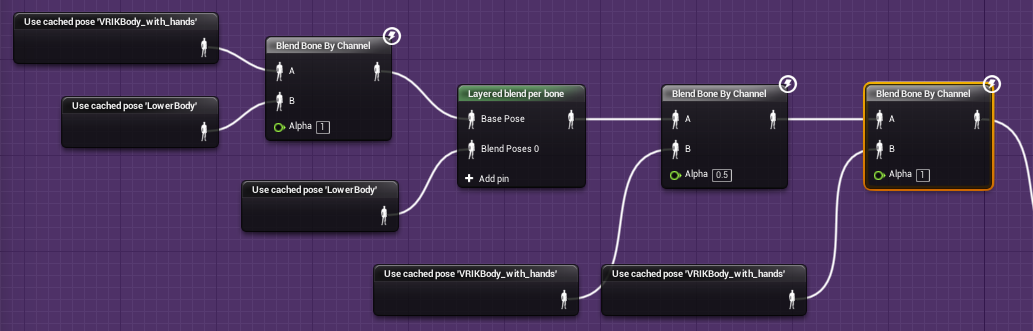


* In Anim Graph apply Torso Rotation to pelvis bone of strafe animation (in world space) to reorient it in a correct way



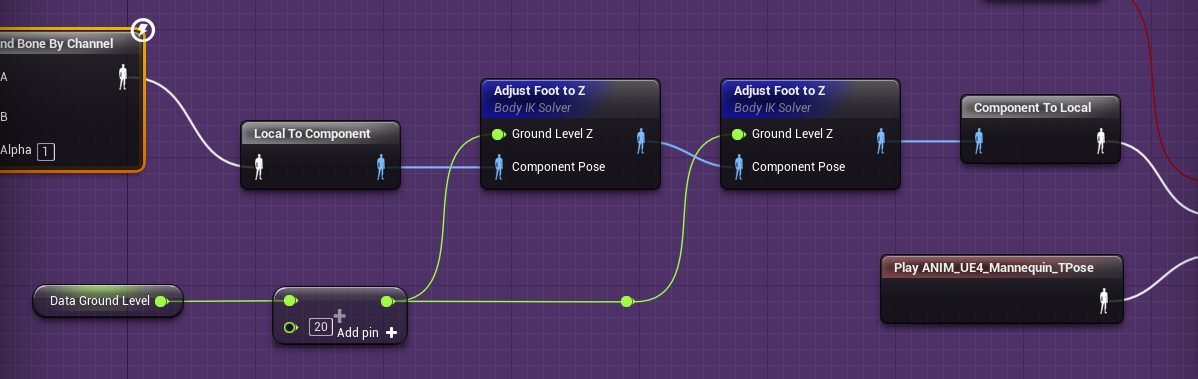
* Then blend strafe animation to PoseSnapshot data. This isn’t simple part. Legs in a strafe animation rely on pelvis rotation and location, so we need to keep at least pelvis rotation to preserve a correct animation. At the other hand, head and hands must be kept at the locations defined in a pose snapshot. If strafe animation change pelvis position, it would create error in hands and head placement. I suggest to keep pelvis location from the pose snapshot and update its rotation from the strafe animation using *BlendBoneByChannel* node. Then, restore upped body in world space from the pose snapshot since the first spine bone after pelvis.





In this setup I have semi-blended first spine bone with fully-restored second spine bone to get more smooth transition. Both *BlendNobeByChannel* nodes here only update rotation in world space.

* And finally, return animated feet on the ground by using custom *AdjustFootToZ* animation node. Ground level Is stored in a structure returned by VRIKBody::GetLastFrameData.



That’s all!

Information

IKBodyData Struct

All transforms are in World Space or relative to Player Pawn origin if both **VRInputFromComponents** and **FollowPawnTransform** flags are false.

|  |  |  |
| --- | --- | --- |
| **MEMBER** | **TYPE** | **DESCRIPTION** |
| Pelvis | *Transform* | Pelvis Transform |
| Ribcage | *Transform* | Ribcage/Spine Transform |
| Neck | *Transform* | Neck Transform |
| Head | *Transform* | Head Transform |
| UpperarmRight\* | *Transform* | Right Upperarm Transform (only if ComputeHandsIK is true) |
| ForearmRight\* | *Transform* | Right Forearm Transform (only if ComputeHandsIK is true) |
| HandRight | *Transform* | Right Palm Transform |
| UpperarmLeft\* | *Transform* | Left Forearm Transform (only if ComputeHandsIK is true) |
| ForearmLeft\* | *Transform* | Left Forearm Transform (only if ComputeHandsIK is true) |
| HandLeft | *Transform* | Left Palm Transform |
| ElbowJointTargetRight | *Transform* | IK Joint Target for right hand in world space (if  ComputeHandsIK is true) |
| ElbowJointTargetLeft | *Transform* | IK Joint Target for left hand in world space (if  ComputeHandsIK is true) |
| ThighRight\* | *Transform* | Right Thigh Transform (only if ComputeFeetIK is true) |
| CalfRight\* | *Transform* | Right Calf Transform (only if ComputeLegsIK is true) |
| ThighLeft\* | *Transform* | Left Thigh Transform (only if ComputeLegsIK is true) |
| CalfLeft\* | *Transform* | Left Calf Transform (only if ComputeLegsIK is true) |
| FootRightCurrent | *Transform* | Right Feet IK instantaneous Transform (only if  ComputeFeetIKTargets is true) |
| FootLeftCurrent | *Transform* | Left Feet IK instantaneous Transform (only if  ComputeFeetIKTargets is true) |
| IsJumping | *bool* | Flag indicates if character is jumping |
| IsSitting\* | *bool* | Flag indicates if character is sitting |
| Velocity | *Vector* | Current Player Vector Velocity. Vector Length is equal to scalar speed (meters per second). |
| GroundLevel | *float* | Current Ground Z Coordinate |

\* - not updated via networking

VR IK Body Component parameters

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **VARIABLE** | |  | **TYPE** | |  | | **DESCRIPTION** |
| **SETUP** | | | | | | | |
| ComputeFeetIKTargets | | *bool* | | |  | | Set this flag to True if you need Feet Transform  Predictions (**FootRightCurrent**, **FootLeftCurrent**) |
| ComputeLegsIK | | *bool* | | |  | | Set this flag to True if you need thigh and calf transforms (**ThighRight**, **CalfRight**, **ThighLeft**, **CalfLeft**). Only works if ComputeFeetIKTargets is true. |
| ComputeHandsIK | | *bool* | | |  | | Set this flag to True if you need upperarm and forearm transforms and joint targets  (**UpperarmRight**, **ForearmRight**, E**lbowJointTargetRight**, **UpperarmLeft**, **ForearmLeft**, **ElbowJointTargetLeft**). |
| UseActorLocationAsFloor | *bool* | | | | | If true, Owning Pawn location Z will be used as floor coordinate. This approach doesn't work properly on slopes and staircases. If false, uses Line Trace to find ground level. | |
| FloorCollisionObjectType | *Collision Channel* | | | | | Collision object type of floor if  UseActorLocationAsFloor is false. | |
| VRInputOption | *VR Input Setup* | | | | | There are three options: Direct input from VR API, Input from scene components (camera and motion controller components), input from external variables updated in tick. If use  components input or networking, call Initialize(...) function on begin play. | |
| LockShouldersRotation | *bool* | | | | | If true, shoulders don't rotate to follow motion controllers location. | |
| FollowPawnTransform | *bool* | | | | | This flag only works if VRInputFromComponents is  false. If true, component uses Pawn Actor  Transform to locate body in world space.  Otherwise it retuns body relative to Pawn Origin. World space calculation is required for unnatural locomotion. | |
| LeftPalmOffset | *Vector* | | | | | Palm location relative to Motion Controller  Component transform (default value is for HTC Vive Motion Controller) | |
| RightPalmOffset |
| **Body Params** | | | | | | | |
| BodyWidth | *float* | | |  | | Y-axis (right-left) body size (modified by body calibration) | |
| HeadHalfWidth | *float* | | |  | | Approximate head radius | |
| HeadHeight | *float* | | |  | | Approximate head height | |
| SpineLength | *float* | | |  | | Approximate distance from pelvis to neck  (modified by body calibration) | |
| HandLength | *float* | | |  | | Approximate distance from collarbone to palm  (modified by body calibration) | |
| FootOffsetToGround | *float* | | |  | | Distance from feet to ground, useful to correct skeletal mesh feet bones Z-offset | |
| NeckToHeadsetOffset | *Vector* | | |  | | Component Space Offset from Neck to Head (X is forward, Z is up), modified by body calibration | |
| RibcageToNeckOffset | *Vector* | | |  | | Component Space Offset from Ribcage to Neck (X is forward, Z is up) | |
| MaxHeadRotation | *float* | | |  | | Max permissible angle between head and pelvis  Yaw rotations (used to correct pelvis rotation) | |
|  | | | | **Networking** | | | |
| ReplicateFullBodyState | *bool* | | |  | | If true, the component calculates body state on local PC and send it to server for other connected users. If false, component sends Head and Hands transforms to server, and every client perform body calculation for each player locally. Choose first option (true) if CPU performance is a priority in your project. Choose remote calculations (false) to optimize a project for network bandwidth. | |
| ReplicateInWorldSpace | *bool* | | |  | | If ReplicateInWorldSpace is set to false (by default), all body data would be converted to actor space before replication and reconverted to world space on remote machines. This operation adds a lot of transforms calculations, but allow to use sliding (Onward-style) locomotion which implies that pawn locations on different PCs are slightly asynchronous at the same moment. Set ReplicateInWorldSpace to true if you don't use sliding player locomotion to optimize CPU usage. | |
| SmoothingInterpolation  Speed | *bool* | | |  | | Speed of interpolation between current and received body state. Set to 0 to disable interpolation. | |

VR IK Body Component Functions

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **RETURN**  **VALUE** | **PARAMETERS** |
| CalibrateBodyAtTPose  CalibrateBodyAtIPose  AutoCalibrateBodyAtPose | *bool* | (no params) |
| Calibrate Body Params at T-Pose (hand to the left and right) and I-Pose (hands down). Calibration will be applied automatically at the second call (order is not important). Returns true if calibration is finished successfully, i.e. after both calls. AutoCalibrateBodyAtPose() detects pose automatically. | | |
| Initialize | *void* | **Camera**: Camera Component Reference  **RightController**: Motion Controller Component  Reference  **LeftController**: Motion Controller Component Reference |
| Call this function on BeginPlay to setup component references if VRInputFromComponents is true | | |
| ActivateInput | *void* | (no params) |
| This function activates VR IK Body component. If you use scene components for data input, use Initialize(…) function instead to initialize component references and activate component. | | |
| DeactivateInput  *void*  (no params) | | |
| The function stops component. It can be reactivated later by Activateinput() call. | | |
| ComputeFrame | *IKBodyData* | **DeltaTime**: float |
| Call in Tick(…) to calculate body state. | | |
| GetLastFrameData | *IKBodyData* | (no params) |
| Returns a last body state struct calculated by ComputeFrame(...) .  function | | |
| ConvertDataToSkeletonFriendly | *IKBodyData* | **WorldSpaceIKBody**: IKBodyData |
| Converts calculated body parts orientation to skeleton bones orientation. It's useful to directly set bone transforms in Anim Blueprint using 'Modify (Transform) Bone' node. Keep in mind that transforms in the returned struct are still in world space. The function doesn’t affects Pelvis. It always have X as forward axis and Z as up. | | |
| ResetTorso | *void* | (no params) |
| Call this function to set Pitch and Roll of Pelvis to zero and Yaw equal to Head Yaw. | | |
| GetCharacterHeight  *float*  (no params) | | |
| Returns calculated or calibrated character from feet to HMD.    height | | |
| GetCharacterLegsLength | *float* | (no params) |
| Returns calculated or calibrated legs length. | | |
| AttachHandToComponent | *bool* | **Hand**: Controller Hand |
|  |  | **Component**: Primitive Component  **SocketName**: Name  **RelativeTransform**: Transform |
| Function attaches hand palm to primitive component. Calculated as relative transform to component’s socket (or relative transform to component itself if socket isn’t specified). Affects Hand Transform. Affects Upperarm/Forearm Transforms and elbow joint target if ComputeHandsIK is true. Returns true if succeed. | | |
| DetachHandFromComponent | *void* | **Hand**: Controller Hand |
| Reattach hand palm to motion controller. | | |
| UpdateInputTransforms *void*  **HeadTransform**: Transform  **RightHandTransform**: Transform  **LeftHandTransform**: Transform | | |
| Call this function in Tick() to update Head and hands transforms if VRInputOption is 'Input from  Variables' | | |
| GetCalibratedBody | *CalibratedBody* | (no params) |
| Returns the struct describing body calibration results. Use it to save and restore body params if you need to respawn player. | | |
| RestoreCalibratedBody | *void* | **BodyState**: CalibratedBody |
| Loads body calibration parameters from CalibratedBody struct. | | |
| ResetCalibrationStatus | *void* | (no params) |
| Mark body as non-calibrated. Function (replicated) keeps existing body params, but allow to recalibrate body if necessary. | | |
| IsBodyCalibrated *bool* (no params) | | |
| Returns true if calibration is complete or calibration data is loaded by RestoreCalibratedBody | | |
| IsValidCalibrationData | *bool* | **BodyParams**: CalibratedBody |
| Check if CalibratedBody variable is valid. Use it if you save and load calibration params. | | |

Events (this feature is not included in the retail version)

|  |  |  |
| --- | --- | --- |
| OnJumpStarted | (no params) | Event called when player starts a jump |
| OnGrounded | (no params) | Event called when player ends a jump |
| ~~OnHeadShake~~ | **Iteration**:  float | Event called when player shakes a head |
| ~~OnHeadNod~~ | **Iteration**:  float | Event called when player nodes a head |
| OnSitDown | (no params) | Event called when player finishing squatting |
| OnStandUp | (no params) | Event called when player stands up |
| OnCalibrationComplete | (no params) | Event called when player body calibration complete successfully |

VR IK Skeletal Mesh Translator

Component extracts data from VR IK Body Component and returns PoseSnapshot object ready-touse with any custom skeletal mesh.

Parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **VARIABLE** |  | **TYPE** | **DESCRIPTION** |
| RootBone, Pelvis, Ribcage,  Head, ShoulderRight,  UpperarmRight,  LowerarmLeft, PalmRight,  ShoulderLeft, UpperarmLeft,  LowerarmLeft, PalmLeft,  ThighRight, CalfRight,  FootRight, ThighLeft,  CalfLeft, FootLeft | *Name* | | Names of bones in a skeleton object. RootBone must be equal to Pelvis (not None) if Pelvis is a root.  Ribcage is the last spine bone. |
| RescaleMesh | *bool* | | Should Component apply player’s height to skeletal mesh scale or not. If true, scale applied on VRIKBody component’s OnCalibrationComplete event. |
| ScaleMultiplierVertical | *float* | | Mesh scale coefficient vertical |
| ScaleMultiplierHorizontal | *float* | | Mesh scale coefficient in a horizontal plane |
| RootMotion | *bool* | | If true, moves both Root bone and Skeletal Mesh Component. |

Functions

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **RETURN VALUE** | **PARAMETERS** |
| Initialize | *bool* | **ControlledSkeletalMesh**: Skeletal  Mesh  Component,  **VRIKBodyComponent**: VR IK Body  Component |
| Initialize references and build information about controlled skeletal mesh. References can be automatically extracted from the owner actor If it has a VRIKBody component and only one Skeletal Mesh Component. Skeletal Mesh Component must contain a Skeletal Mesh object in T Pose in a moment of initialization. Returns true if initialization was successful. | | |
| IsInitialized | *bool* | (no params) |
| Is component initialized or not? | | |
| GetSkeletalMeshSetup | *FSkeletalMeshSetup* | (no params) |
| Get current skeletal mesh bones setup. Use this function to save/restore mesh data without reinitialization. | | |
| RestoreSkeletalMeshSetup  *SkeletalMeshSetup*  **SkeletalMeshSetup**: FSkeletalMeshSetup | | |
| Load skeletal mesh bones setup to component. Use this function to save/restore mesh data without reinitialization. | | |
| GetLastPose | *bool, PoseSnapshot* | (no params) |
| Get current pose from VRIKBody component adjusted for controlled skeletal mesh. Returns PoseSnapshot object and true if the component is initialized and calibrated. | | |
| GetMeshWorldTransform | *Vector, Rotator* | (no params) |
| Return current Location and Rotation of Skeletal Mesh Component in World Space if Root Motion is enabled or predicted Location and rotation if Root Motion is disabled. | | |
| UpdateSkeleton | *-* | (no params) |
| Must be called in Tick(…) after VRIKBody::CalculateFrame and before any artificial locomotion to load calculated data and update skeletal mesh position (if Root Motion is enabled) | | |
| ForceSetComponentAsCalibrated | *-* | (no params) |
| Function marks body as calibrated and allow to receive input from VRIKBody component. Note:  function isn't replicated, call it on all clients manually. | | |