ANYIK GUIDE
(v 2.0r8)

1. Upgrade Notes for Earlier Versions (1.x)

Version 2.0 introduces new scripts with new names for every type of IK script, although old scripts (if kept) may still work, it is strongly recommended to remove AnyIK folder completely before downloading and importing AnyIK 2.0. You will need to remove old AnyIK scripts from your GameObjects and add new scripts following this guide.

We are aware that it could be frustrating to remove all the scripts and re-add them and apologize for it. But we assure you it was not possible to cleanly upgrade scripts in 1.x version while adding the feature that enables all the scripts to work simultaneously and accurately.

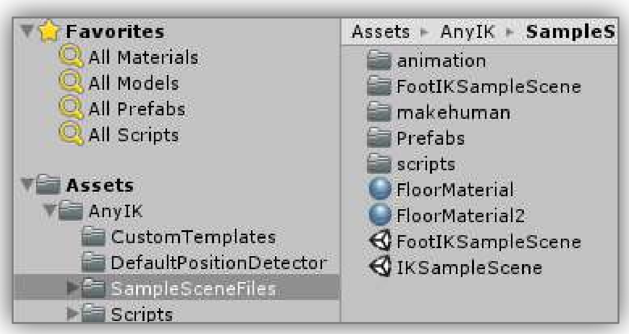
Also with version 2.0 and forward we support (and require) only Animator component. Our focus will be on humanoid characters but we will support non-humanoid characters as long as they have an Animator component. (With v.2.0r2 an Animator relay script added which enables users to have AnyIK scripts on one GameObject and the Animator component in another GameObject. This is especially useful for dynamically created avatars which is not available on scene at design time like UMA)

2. Introduction

AnyIK provides scripts that enables you to define IK chains (AnyIKChainController) by the character's rig's bones. Animator component is required but model can be humanoid or not. When the model is humanoid, it is automatically detected by script and many of the settings will be handled for you.

AnyIK also provides seperate scripts for head look IK (AnyIKHeadLookController) and foot placement IK (AnyIKFootPlacementController) operations. A character may only have one foot placement script and one head look script but it may have as many AnyIKChainController scripts as you need. With version 2.0 and up, scripts effect common bones of multiple chains without causing inaccuracy to IK chains that has been processed before it. In other words, you may have a chain from right hand to hip bone and another chain from left hand to hip bone, they will work and find a suitable rotation for common bones (i.e. spine and chest) so both hands can reach or stop as close as possible to their respective targets.

This coordination is achieved by AnyIKBoneStructureCoordinator script that is automatically added when any of the IK script is added to character. This script is very important as it also defines the bone structure and bone limits by predefined templates. Currently MCS/DAZ, Makehuman, Robot Kyle, iClone G6 and UMA bone structure templates are available. As for bone limits, Unity default bone limit template and one slightly customized bone limit template is available. You may also add your own custom templates. (See AnyIKBoneStructureCoordinator section for details.). With version 2.0r3 an option added to bone structure coordinator namely "Do not use bone structure template". (This option will enable you to use a model which is not in the list, quickly without defining a template) Read more in BoneStructureCoordinator section.



When you imported AnyIK into your project you'll see a folder structure like this.

In the SampleSceneFiles folder there are two scenes. These scenes are provided so you can have an idea on script settings and have a working sample to follow.

First scene demonstrates foot IK setup. When you run this scene, you'll be able to point and click around to move the character. The character is a Makehuman character with high heels rigged to foot bones. The animations used are not for high heels, AnyIK is handling that part (you can see the difference by disabling the heels option.)

The other scene demonstrates head look and IK chains. Defined chains have a few common bones (chest and spine namely). You can move around the target game objects and see how the character reacts.

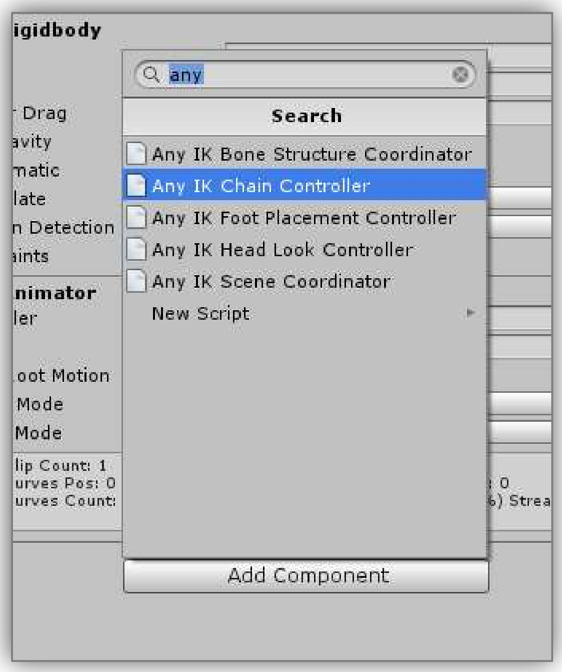
The following section will explain all the settings on AnyIK scripts, so you can setup the scripts from scratch.

You may check out the [tutorial page]() for an online and up-to-date version of this guide or [product page]() for videos.

3. AnyIK Scripts

a. AnyIKChainController

This script is used to define an arbitrary IK chain starting at any bone. You may add/remove bones to chain. You may define as many IK Chain as you like by simply adding AnyIKChainController script to your character. To add AnyIKChainController script click Add Component on your character's inspector window, type "any" and select Any IK Chain Controller from the list.



This will add both AnyIKChainController and AnyIKBoneStructureCoordinator to the GameObject. Default values will be set and you will need to select Limit Preset and Bone Structure Preset that suits your model.

The initial pose bone rotations differ from model to model so for best results a preset file is needed to store this information for each model. A few presets are provided by AnyIK and it is possible to add a custom template. (See AnyIKBoneStructureCoordinator section for details.)

Initially AnyIKChainController script is added to GameObject with no target and no bones defined. Other properties are set to defaults.

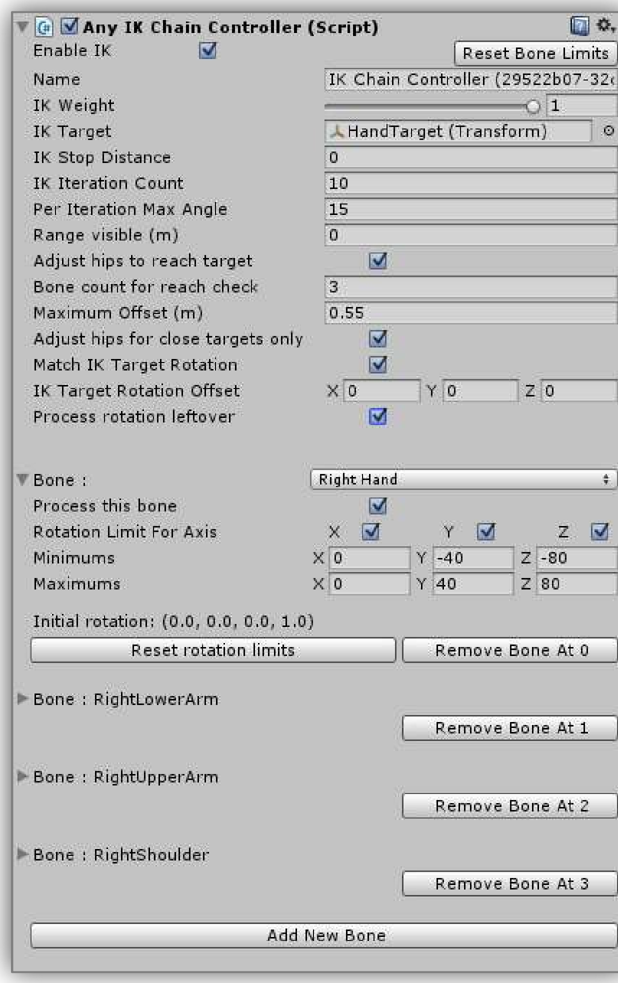
Keep in mind that you need at least 2 bones in the chain to see any movement.

Enable IK, turns this IK chain on and off. Reset Bone Limits button resets minimum and maximum bone limits for all bones defined in this script per the defined bone limit template.

Name is randomly given when script is added, and can be changed afterwards. This name is for distinguishing between added scripts and doesn't have any other use.

IK weight is the weight of the IK movement. If it is 1 then IK is fully processed, if it is 0 IK is not processed at all.

You can drag a game object to IK Target. This GameObject's position will be where IK chain targets.



IK stop distance is where script will consider target reached. It will not process next IK iteration if target is in this range (unit is meters).

IK iteration count is the limit to how many times the IK process will iterate. If target is achieved (target is closer than defined stop distance) on any iteration it will stop, if number of iterations defined by this property are processed and target is not achieved, it will stop at current distance.

Per iteration max angle defines how much each bone is rotated every iteration. Keep in mind that too small angles may cause inaccurate targeting.

Range visible defines the range in meters where the script will be considered disabled. If this is 0 script will be always enabled. If non-zero then when camera is further than this amount (in meters), script will be considered disabled. This option will be used with Filter IK By Camera Distance option in AnyIKBoneStructureCoordinator script. If that option is unchecked range defined here will be ignored, IK will be processed.

Adjust hips to reach target option, when selected, changes vertical hips position downwards when target is not reachable. Bone count for reach check is used to calculate the reach. Maximum offset is the limit on how much the hips can be moved down. Adjust hips for close targets only option restricts hips movement so that it is processed only when target is in close proximity.

If Match IK Target Rotation is checked IK chain's first bone will try to match the rotation of target transform. This rotation will be offset by the angles defined by IK target rotation offset value (visible when selected).

When Process rotation leftover is selected, if target rotation cannot be matched at the end effector bone, leftover rotation (on bone's direction axis) is passed to next bone.

You can add new bones to the IK chain by clicking the "Add New Bone" button at the bottom.

You can select the first bone from the combo box if your character has a humanoid animator. If not, then you'll be able to drag your bone from hierarchy view to inspector. After first bone, when you add a new bone to the chain, parent bone is automatically added to the chain.

If you are using humanoid bones, default rotation restrictions will be set per the limit preset defined in AnyIKBoneStructureCoordinator script.

If your character’s animator is not humanoid default limits will be wide enough so there will be no limits. You can always change rotation limits. Also, you may opt not to apply limits by unchecking the axis above the limits.

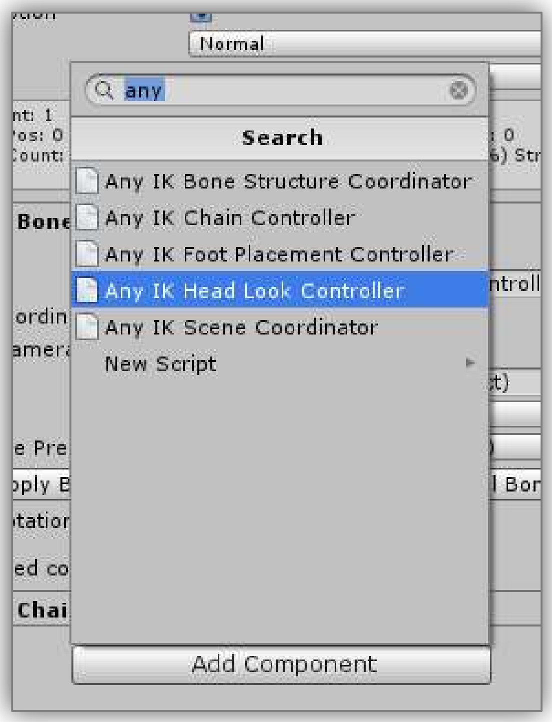
If you uncheck Process this bone option, no IK will be applied to that bone which will be practically skipped.

You may add as many bones as you like to the IK chain, but eventually you will be adding hips (or parent game object of the hips). This is not recommended as it will rotate these game objects and may lead to unexpected behavior.

You may add multiple AnyIKChainController scripts to the same character. Every chain is processed simultaneously. Common bones are processed mutually by chains and rotated considering each chain affecting it.

b. AnyIKHeadLookController

Head controller script handles head and eye “look at” function. More bones can be added to head bone (i.e. neck, chest, spine)



You may add only one head look script to a GameObject by simply adding AnyIKHeadLookController script to your character. To add AnyIKHeadLookController script click Add Component on your character’s inspector window, type “any” and select Any IK Head Look Controller from the list.

This will add both AnyIKHeadLookController and AnyIKBoneStructureCoordinator to the GameObject. Default values will be set and you will need to select Limit Preset and Bone Structure Preset that suits your model.

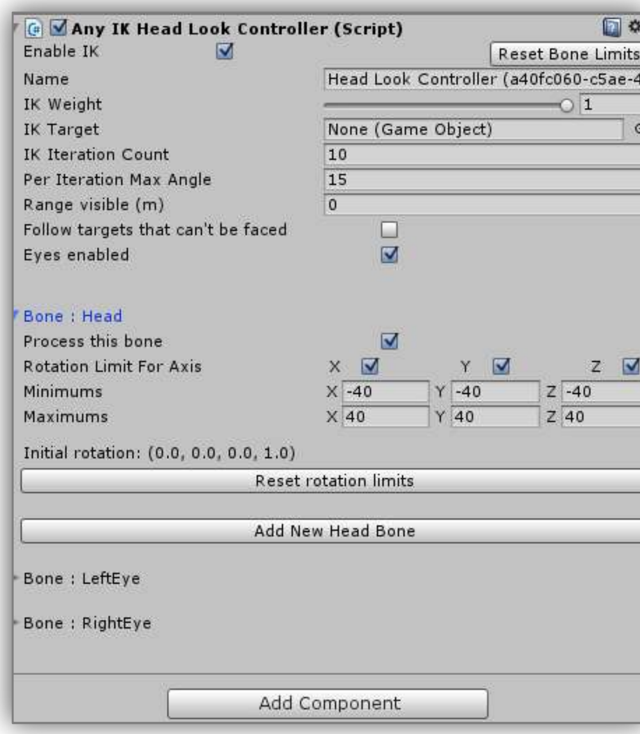
The initial pose bone rotations differ from model to model so for best results a preset file is needed to store this information for each model. A few presets are provided by AnyIK and it is possible to add a custom template. (See AnyIKBoneStructureCoordinator section for details.)

Enable IK, turns this IK chain on and off. Reset Bone Limits button resets minimum and maximum bone limits for all bones defined in this script per the defined bone limit template.

Name is randomly given when script is added, and can be changed afterwards. This name is for distinguishing between added scripts and doesn’t have any other use.

IK weight is the weight of the IK movement. If it is 1 then IK is fully processed, if it is 0 IK is not processed at all.

IK target can be a regular game object in which case the target position is game object's position, or it can be a humanoid (target game object should have a humanoid animator). If target has a humanoid animator Use Humanoid Target option becomes visible. If selected you'll have an option to select which humanoid bone to look at (i.e. Head, Right Hand, Left Foot etc.)



IK iteration count is the limit to how many times the IK process will iterate. If target is achieved (target is closer than defined stop distance) on any iteration it will stop, if number of iterations defined by this property are processed and target is not achieved, it will stop at current distance.

Per iteration max angle defines how much each bone is rotated every iteration. Keep in mind that too small angles may cause inaccurate targeting.

Range visible defines the range in meters where the script will be considered disabled. If this is 0 script will be always enabled. If non-zero then when camera is further than this amount in meters, script will be considered disabled. This option will be used with Filter IK By Camera Distance option in AnyIKBoneStructureCoordinator script. If that option is unchecked, range defined here will be ignored, IK will be processed.

When follow target that can't be faced option is selected head (and neck and eyes too) rotation stays at the limits when target rotation cannot be achieved. When this is not selected, head turns back to animation when rotation cannot be achieved.

Follow tolerance angle is the tolerance after the point that character cannot face the target anymore. When the tolerance is exceeded head look weight is lowered down to 0. This option is available when follow target that can't be faced option is not selected.

When eyes enabled option is selected, defined eyes will follow IK target.

Head is automatically added to IK chain (you need to define head bone transform if animator is not humanoid). Additional bones can be added to chain. Whenever you click "Add New Head Bone" button, parent bone of the current last bone will be added to chain (it goes like Head, Neck, Chest etc.)

c. AnyIKFootPlacementControler

Foot placement script handles foot placement of the character. It matches the ground slope / height and adjusts the feet position and hip height accordingly. It also optionally rotates the feet and toes to modify the animation for high heel.

You may add only one foot placement script to a GameObject by simply adding AnyIKFootPlacementController script to your character. To add AnyIKFootPlacementController script click Add Component on your character's inspector window, type "any" and select Any IK Foot Placement Controller from the list.

This will add both AnyIKHeadLookController and AnyIKBoneStructureCoordinator to the GameObject. Default values will be set and you will need to select Limit Preset and Bone Structure Preset that suits your model.

The initial pose bone rotations differ from model to model so for best results a preset file is needed to store this information for each model. A few presets are provided by AnyIK and it is possible to add a custom template. (See AnyIKBoneStructureCoordinator section for details.)

Enable IK, turns this IK chain on and off. Reset Bone Limits button resets minimum and maximum bone limits for all bones defined in this script per the defined bone limit template.

Name is randomly given when script is added, and can be changed afterwards. This name is for distinguishing between added scripts and doesn't have any other use.

If adjust hips height by lowest foot is checked, hip is moved down by the amount from lowest foot to ground level. So, every foot's target point on the ground can be reached. (recommended)

If consider front and back ends of foot is checked, two more raycast are executed from both ends of the foot. Highest y value is used, so for example if only the tip of the foot is into the step, it moves target point to step's level. When this option is not selected, more clipping will occur with stair like floor.

Hips vertical offset is applied to hips bone and moves all bones up or down. It's for fine tuning the animation height (for animations with too high or too low feet ground distance).

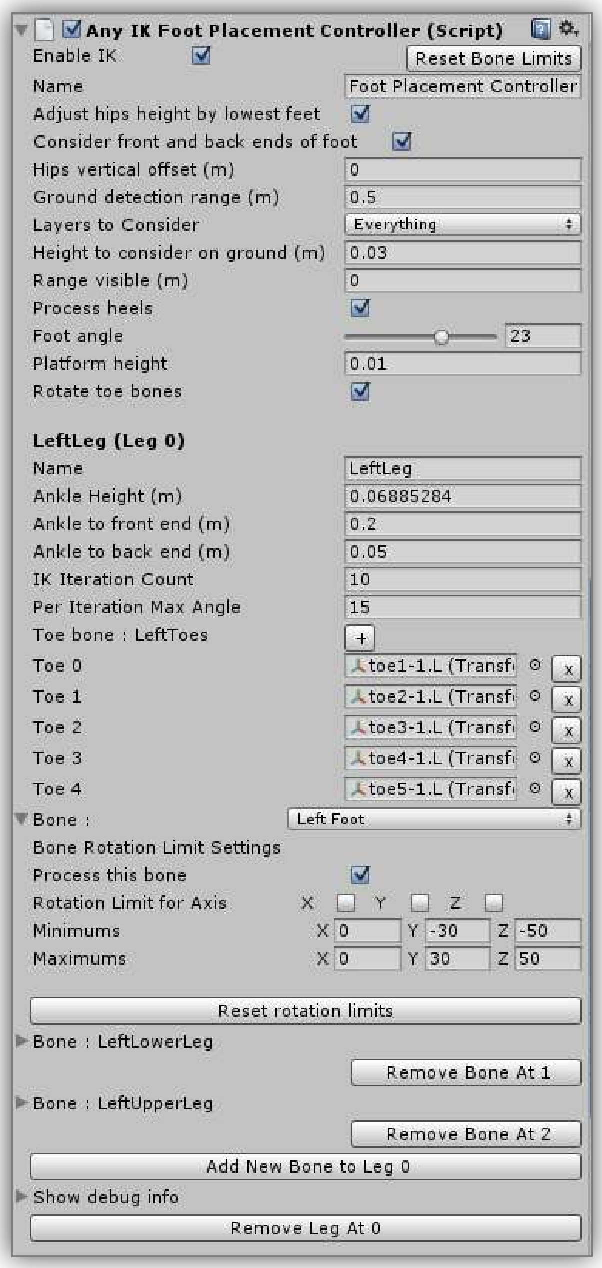
Ground detection range affects the length of the raycasts. A raycast is cast from a point this range higher than the foot position with raycast size of two times this range. In other words, this range above and this range below the feet is checked for ground/floor.

With layers to consider combobox you can select which layers are considered as floor. Default is everything.

Height to consider on ground is the sensitivity of the on-ground check. It may be adjusted to fit the animation. This is the distance the foot is considered fully on ground when further than this, leg IK weight is reduced by distance.

Range visible defines the range in meters where the script will be considered disabled. If this is 0 script will be always enabled. If non-zero then when camera is further than this amount in meters, script will be considered disabled. This option will be used with Filter IK By Camera

Distance option in AnyIKBoneStructureCoordinator script. If that option is unchecked, range defined here will be ignored, IK will be processed.



Process heels option rotates the feet and raises whole model by heel height to make foot animation compatible with high heels (rigged to foot bone fitting the model feet). This option is to use the same walk / run animations for heels. Angle and platform height can be adjusted. You can decide if toes should be rotated or not (usage depends on how tip of the shoes are rigged)

You also have legs and leg bones. If you animator is humanoid, then left and right leg is created with 3 bones each forming the legs. If not, you can add and remove legs and leg bones for each leg.

In addition to bones and rotation limits legs have following properties:

Ankle height : Foot bone is usually at ankle (at least it is with humanoid bones). So IK target point must be higher than ground level by ankle height so that foot sits on floor. This property is essential so that the feet sit on the ground perfectly. (This property is autodetected)

Ankle to front end : This option is used to calculate raycast point from front end of the feet. (if consider front and back ends of foot option is not selected this is not used)

Ankle to back end : This option is used to calculate raycast point from back end of the feet. (if consider front and back ends of foot option is not selected this is not used)

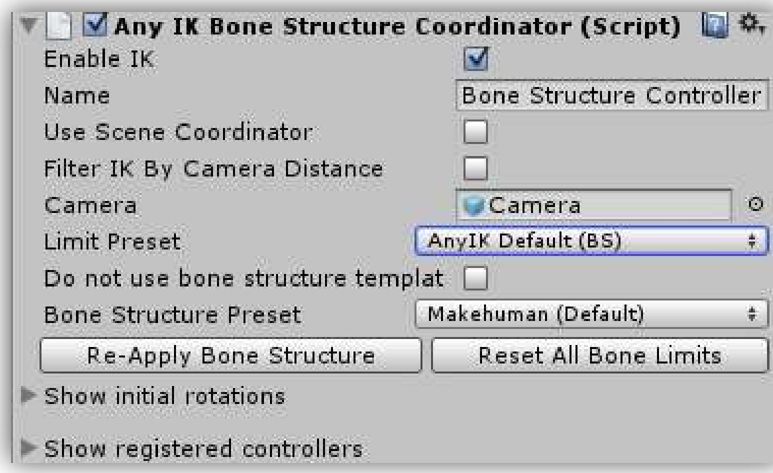
IK iteration count is the limit to how many times the IK process will iterate. If target is achieved (target is closer than defined stop distance) on any iteration it will stop, if this many iterations are processed and target is not achieved, it will stop (at current distance).

Per iteration max angle defines how much each bone is rotated every iteration. Keep in mind that too small angles may cause inaccurate targeting.

Toe bone is detected when animator is humanoid. If your model’s toe bones are connected to separate toe bone then it is enough. Some models have 5 toe bones connected directly to foot bone. Script autodetects the bones connected to foot and populate toe bone list. (These are rotated per the rotate toe bones option for heels.)

d. AnyIKBoneStructureCoordinator

As mentioned before when an AnyIK script is added to a GameObject, the bone structure coordinator script is added automatically.



Main purpose of this script is holding the definition of the bone structure and default bone limits. Every model has a different orientation of the bones. Because of this a preset is used to hold this information. Currently MCS/DAZ, Makehuman, Robot Kyle, iCloneG6 and UMA presets are available. If your model doesn't conform to one of these, you may create a custom preset and use it. Also with version 2.0r3 an option added to bone structure coordinator namely "Do not use bone structure template". (This option will enable you to use a model which is not in the list, quickly without defining a template) If this is selected the reference pose will not be the Unity muscle pose but the initial pose of the model (which can be a T-Pose, A-Pose etc.). It is detected at game/scene start and used as reference. When this option is selected a compatible bone limit template must also be selected. Bone limits now have a "BS" (bone structure) or "No BS" (no bone structure) indication next to their names. (Newly added "T-Pose (No BS)" bone limit template is a sample template roughly compatible to Makehuman model. Limits may still apply to wrong axes. So, either change the limits on inspector for each IK script or create a custom bone limit template applying to your model and use the reset limits buttons. The latter is recommended.)

The bone limit preset holds the default constraints of the bones. Default uses unity default limits, "AnyIk Default" option is slightly modified unity defaults for better foot IK results.

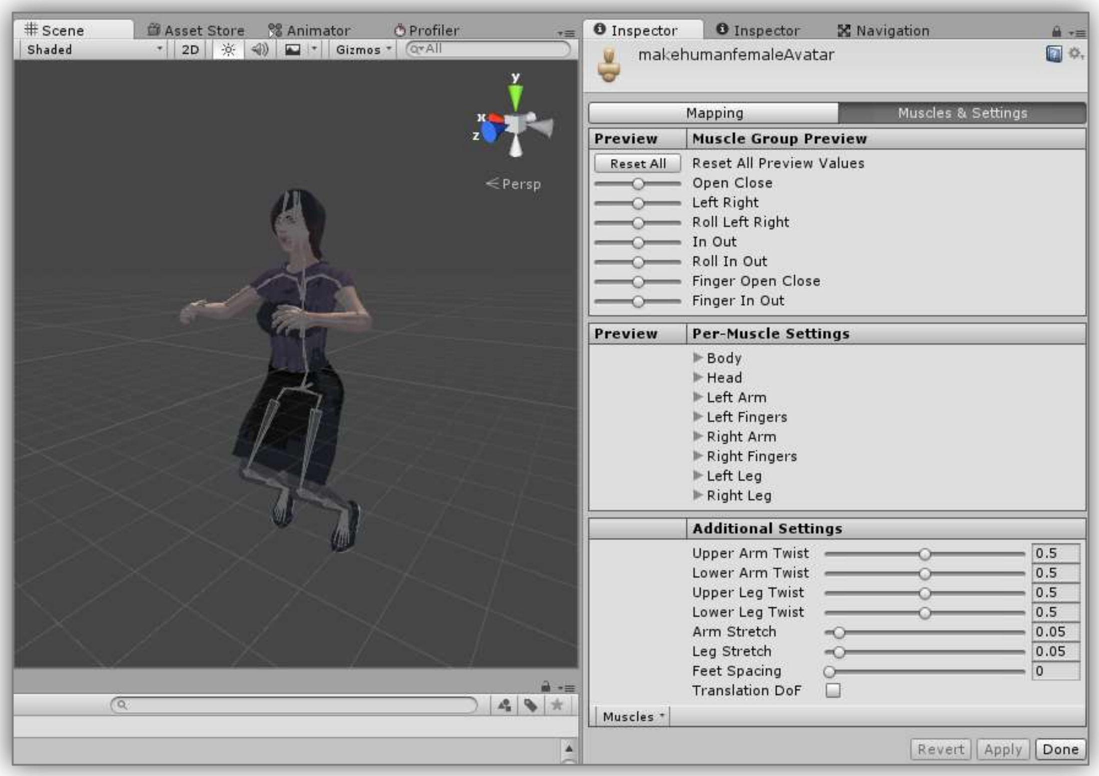
When you select your model, and hit configure button at the rig tab mapping settings will open. Select Muscles and Settings tab to see Unity defaults.

You can see limits for every bone. Limits would be named differently but order of the limit will be z, y, x. (Please note fully constrained axis will be skipped in this window)

This muscle settings window will be useful for creating custom bone structure presets. Because per the orientation of the bone axis names might be switched (for example z rotates x). This will make the bone limits affect wrong axis. In presets, for every bone there is a definition of axis mapping.

Also, bone structure presets hold the initial pose rotations of each bone. This is needed to process limits properly. The initial pose is considered the pose in muscles and settings window.

A tool to generate the initial rotations for a given model is provided by AnyIK package. Go to DefaultPositionDetector folder and open DefaultPositionDetectorScene. Drag your model as the child of "DefaultPositionDetector" GameObject in the scene. Set the animator's avatar property to your model's avatar on DefaultPositionDetector.



When you hit play you should see that your model changes pose to initial pose as in muscles and settings page.

Now go to the console and copy the content of the log to some text file (you should select the log line and copy the content from detail below). This text will be the content of GetDefaultPoseLocalRotations() method of the custom preset script file.

Go to AnyIK -> CustomTemplates folder and duplicate (ctrl d) the SampleCustomBoneStructureTemplate file. Rename it to your liking and open it by a c# editor. (Remember to set the class name in content to whatever you set as the file name) Replace the contents of GetDefaultPoseLocalRotations() method by the text we created earlier (remember the clean unity log's last few lines that is automatically added by unity).

Save the file. Now your preset is half ready. You need to set the content of GetAxisConvertedVectorForHumanBone method too. Unfortunately, this is a manual process. You need to set map every axis of every bone. As I said earlier musles window will guide you. If you skip this, IK will work but bone limits will effect wrong axis mostly.

When your custom preset is ready select Custom option from Limit Preset. This will reveal a text box where you can type the name of your custom preset file. When you type the name (exactly) and hit enter it should replace the class name with fully qualified name of it. Your preset is in use now.

Same routine is used when creating bone limit templates.

***Note** : If you don't want to rework GetAxisConvertedVectorForHumanBone method of the bone structure preset you can leave it as is and instead create a limit preset that works with that mapping. This is not ideal but should work.*

e. AnyIKSceneCoordinator

This script is most of the time not needed at all. All the coordination between scripts of a character is done by AnyIKBoneStructureCoordinator already.

But sometimes two characters need to use IK interactively with each other (like one character reaches to another's hand while that other character's hand reaches for something else).

In this case scene coordinator processes these characters' IK scripts together. This will force IK movements accuracy.

This script should be used when this rare need arises only.

To enable this put AnyIKSceneCoordinator script on an empty GameObject in the scene. Select Use scene coordinator option on the character's AnyIKBoneStructureCoordinator in the inspector (scene coordinator property will be revealed). Drag GameObject with scene coordinator to scene coordinator property field.

f. AnyIKAnimatorRelay

This script enables user to have AnyIK scripts on one GameObject and Animator of the character on another GameObject. It is recommended to have AnyIK scripts on where your character's Animator is but sometimes it may not be possible.

For example UMA creates the character dynamically at runtime and it's Animator is not available until character is initialized at runtime.

Since animator may not be available until runtime, when this script is added to the GameObject, AnyIK scripts will use the is humanoid value on this script.

Is runtime created option is used to define if Animator component will be available at runtime.

If Animator is on another GameObject (not created at runtime) you may drag the game object containing the animator to "Animator" field. If Animator is not available skip this one.

GameObject with Animator can be used similarly if game object that will hold the Animator is available but Animator itself will be available at runtime. If not skip this one too.

If "Try to Find GameObject By Name" is checked when game is started script will search the indicated name and get the Animator component on it. For example, the name of the game object can be defined when an UMA character is created. (of course, the name must be unique)

If you don't have the name beforehand or name is not unique there is one other way to set the Animator. You may get the AnyIKAnimatorRelay script and use the SetAnimator(animator) method on it. This should initialize all AnyIK scripts with animator.

Following is a sample code for UMA but the same technique could be for other models that are created at runtime. The relevant code that you would need to add is marked.

```
using UnityEngine;
using UMA;
using AnyIKLibrary;

public class UMAMaker1 : MonoBehaviour
{
    public UMAGeneratorBase generator;
    public SlotLibrary slotLibrary;
    public OverlayLibrary overlayLibrary;
    public RaceLibrary raceLibrary;
    public RuntimeAnimatorController animController;
    private AnyIKAnimatorRelay animRelay;
    private UMADynamicAvatar umaDynamicAvatar;
    private UMAData umaData;
    private UMADnaHumanoid umaDna;
    private UMADnaTutorial umaTutorialDNA;
    private int numberOfSlots = 20;
    void GenerateUMA() {
            GameObject go = new GameObject("MyUMA");
            umaDynamicAvatar = go.AddComponent<UMADynamicAvatar>();
            umaDynamicAvatar.Initialize();
            umaData = umaDynamicAvatar.umaData;
            umaDynamicAvatar.umaGenerator = generator;
            umaData.umaGenerator = generator;
            umaData.umaRecipe.slotDataList = new SlotData[numberOfSlots];
            umaDna = new UMADnaHumanoid();
            umaTutorialDNA = new UMADnaTutorial();
            umaData.umaRecipe.AddDna(umaDna);
            umaData.umaRecipe.AddDna(umaTutorialDNA);
            CreateMale();
            umaDynamicAvatar.animationController = animController;
            umaDynamicAvatar.UpdateNewRace();
            go.transform.parent = this.gameObject.transform;
            go.transform.localPosition = Vector3.zero;
            go.transform.localRotation = Quaternion.identity;
   }
   void CreateMale() {
            var umaRecipe = umaDynamicAvatar.umaData.umaRecipe;
            umaRecipe.SetRace(raceLibrary.GetRace("HumanMale"));
            umaData.umaRecipe.slotDataList[0] = slotLibrary.InstantiateSlot("MaleEyes");
            umaData.umaRecipe.slotDataList[0].AddOverlay(overlayLibrary.InstantiateOverlay("EyeOverlay"));
            umaData.umaRecipe.slotDataList[1] = slotLibrary.InstantiateSlot("MaleInnerMouth");
            umaData.umaRecipe.slotDataList[1].AddOverlay(overlayLibrary.InstantiateOverlay("InnerMouth"));
            umaData.umaRecipe.slotDataList[2] = slotLibrary.InstantiateSlot("MaleFace");
            umaData.umaRecipe.slotDataList[2].AddOverlay(overlayLibrary.InstantiateOverlay("MaleHead02"));
            umaData.umaRecipe.slotDataList[3] = slotLibrary.InstantiateSlot("MaleTorso");
            umaData.umaRecipe.slotDataList[3].AddOverlay(overlayLibrary.InstantiateOverlay("MaleBody02"));
            umaData.umaRecipe.slotDataList[4] = slotLibrary.InstantiateSlot("MaleHands");
            umaData.umaRecipe.slotDataList[4].AddOverlay(overlayLibrary.InstantiateOverlay("MaleBody02"));
            umaData.umaRecipe.slotDataList[5] = slotLibrary.InstantiateSlot("MaleLegs");
            umaData.umaRecipe.slotDataList[5].AddOverlay(overlayLibrary.InstantiateOverlay("MaleBody02"));
            umaData.umaRecipe.slotDataList[6] = slotLibrary.InstantiateSlot("MaleFeet");
            umaData.umaRecipe.slotDataList[6].AddOverlay(overlayLibrary.InstantiateOverlay("MaleBody02"));
            umaData.umaRecipe.slotDataList[3].AddOverlay(overlayLibrary.InstantiateOverlay("MaleUnderwear01"));
            umaData.umaRecipe.slotDataList[5].AddOverlay(overlayLibrary.InstantiateOverlay("MaleUnderwear01"));

    }
    void Start()
    {
            animRelay = GetComponent<AnyIKAnimatorRelay>();
            GenerateUMA();
    }
    void LateUpdate()
    {
            if (animRelay.GetAnimator() == null)
                    animRelay.SetAnimator(umaDynamicAvatar.gameObject.GetComponent<Animator>());
    }
}
```