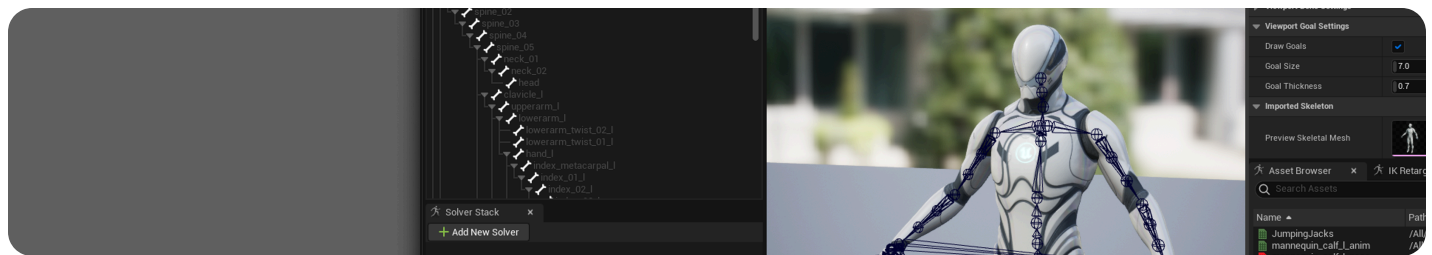


Developer  
/ Documentation  
/ Unreal Engine ▾  
/ Unreal Engine 5.4 Documentation  
/ Animating Characters and Objects  
/ Skeletal Mesh Animation System  
/ Animation Assets and Features  
/ IK Rig  
/ IK Rig Editor  
/ Using Python with IK Rigs

# Using Python with IK Rigs

Use Python scripting to create and edit IK Rigs to automate workflows.



When creating and using [IK Rigs](#) to animate characters and objects in **Unreal Engine**, you can use custom [Python scripting](#) to control and automate IK Rigs asset workflows.

This document provides an overview and example python scripts you can reference and use to edit and interface with IK Rig assets.

## Overview

An IK Rig is a set of controls and solvers that can apply motion data to a skeletal mesh asset in Unreal Engine. IK Rigs consists of multiple parts:

- A Hierarchy of **bones**, **goals**, **goal settings**, and **bone settings**.
- A list of [IK Solvers](#) that influence the Hierarchy's bones and their relationship to one another.
  - Goals are used to drive the IK Solvers, while goal settings and bone settings are used to define the Solver's behaviors.
- A **Retarget Root** and a list of **Retarget Chains** used for [IK Retargeting](#).

- A preview scene used exclusively for an in editor preview.

The IK Rig system is constructed using a Model > View > Controller design formula. The data model itself using the following terminology:

- **Solver:** Which contains the **Inverse Kinematic** solution to rotating and positioning bones in a chain. Multiple Solvers can also be used at the same time to further customize the effect of the IK on the final pose.
- **Goal:** Serves as the **effector point** for your IK chains. Goals are used in conjunction with Solvers to modify an incoming pose to reach the goal locations. Goals can be assigned to multiple solvers.
- **Goal Settings:** Solver specific settings that exist within the IK goal. For each solver assignment to a Goal, there is a different solver specific settings object.
- **Bone Settings:** Solver specific settings that exist within the Bone. For each solver assignment to a Bone, there is a different solver specific settings object.
- **Retarget Root:** The root of the character motion that would be used to transfer from a source to a target.
- **Retarget Chains:** A chain of bones that are defined in order to transfer the motion from a source to a target.

## Accessing the IK Rig

The first step when scripting against any IK Rig is to gain access to the main object you'll interact with, or set the `The IKRigDefinition`. There are several ways to do that depending on your situation. The following examples will illustrate how you can define the IK Rig object using Python scripts.

## Loading an existing IKR asset

To access an existing IK Rig asset (`ikr`) you can simply load the asset using the following example script:

```
1
2 import unreal
3
```

```
4 # Ensure the file path is correct for the location of your asset in your
   project.
5
6 ikr = unreal.load_asset(name =
   '/Game/Characters/Mannequins/Rigs/IK_Mannequin', outer = None)
7
```

 Copy full snippet

## Create a new IKR asset

To create a new IK Rig asset you can use the following factory:

```
1
2 # Get the asset tools.
3
4 asset_tools = unreal.AssetToolsHelpers.get_asset_tools()
5
6 # Create an IK Rig in the location defined by the file path. For example: `
   .../Game/IK_Mannequin`.
7
8 ikr = asset_tools.create_asset(asset_name='IK_Mannequin',
9
10 package_path='/Game/', asset_class=unreal.IKRigDefinition,
11
12 factory=unreal.IKRigDefinitionFactory())
13
```

 Copy full snippet

## Editing an IK Rig

### Preparing for an Edit by Accessing the Controller

The controller is the central object you can use to make any change to the IK Rig. The controller has many functions that you can use to make edits. The following examples are a subset of methods you can use Python scripting to make edits to IK Rig assets:

```
1
2 # Get the IK Rig controller.
3
4 ikr_controller = unreal.IKRigController.get_controller(ikr)
5
```

 Copy full snippet

## Assigning a Skeletal Mesh in a new IK Rig

When creating a new IK Rig, you will need to assign it a skeletal mesh. You can use the following Python scripts to load, assign, a skeletal mesh asset to an IK Rig as well as reference, or get a Skeletal Mesh asset assigned to an IK Rig.

```
1
2 # Load the Skeletal Mesh asset.
3
4 skel_mesh = unreal.load_asset(name =
    '/Game/Characters/Mannequins/Meshes/SKM_Manny_Simple')
5
6 # Assign Skeletal Mesh asset to the IK Rig.
7
8 ikr_controller.set_skeletal_mesh(skel_mesh)
9
10 # Get the Skeletal Mesh assigned to the IK Rig.
11
12 set_mesh = ikr_controller.get_skeletal_mesh()
13
```

 Copy full snippet

## Auto Retarget Chains and Auto FBK

From the controller, you can apply the commands to automatically generate Retarget Chains and a Full Body IK solver with goals. The auto generation is based on commonly known bipedal skeletons.

```
1 # Apply auto generated retarget chains from commonly known biped skeletons
2 ikr_controller.apply_auto_generated_retarget_definition()
3
4
5 # Apply auto generated Full Body IK solver from commonly known biped
  skeletons
6 ikr_controller.apply_auto_fbik()
```

 Copy full snippet

## Adding, Editing, and Querying IK Solvers

You can easily add different IK Solvers via the controller. Adding a solver requires specifying the **IK Rig Solver** class. This is required since there can be custom IK Rig solvers that are used in a project. As long as the base class is a `IKRigSolver`, any class input will work.

```
1
2 # Add a FBIK, Body Mover, Limb, Pole, and Set Transform Solver to the IK
  Rig.
3
4 fbik_index = ikr_controller.add_solver(unreal.IKRigFBIKSolver)
5
6 bodymover_index = ikr_controller.add_solver(unreal.IKRig_BodyMover)
7
8 limb_index = ikr_controller.add_solver(unreal.IKRig_LimbSolver)
9
10 pole_index = ikr_controller.add_solver(unreal.IKRig_PoleSolver)
11
12 settransform_index = ikr_controller.add_solver(unreal.IKRig_SetTransform)
13
```

 Copy full snippet

When adding a solver, the controller will output a solver index. This index determines the solver order of which the IK Rig will solve. You can use the Index to toggle the solver's operation state, move solvers in the stack, or remove a specific solver.

```
2 # Enable or Disable a specific solver.
3
4 ikr_controller.set_solver_enabled(bodymover_index, False)
5
6 # Move a solver.
7
8 ikr_controller.move_solver_in_stack(bodymover_index, limb_index)
9
10 # Remove a solver.
11
12 ikr_controller.remove_solver(bodymover_index)
13
14 ikr_controller.remove_solver(limb_index)
15
16 ikr_controller.remove_solver(pole_index)
17
18 ikr_controller.remove_solver(settransform_index)
19
```

 Copy full snippet

You can also query for the solver index or for all solvers in the IK Rig using the following scripts:

```
1
2 # Get the IK Rig Controller's first solver in the Index.
3
4 fbik_solver = ikr_controller.get_solver_at_index(fbik_index)
5
6 # Getting how many solvers associated with the IK Rig Controller.
7
8 num_solvers = ikr_controller.get_num_solvers()
9
10
```

 Copy full snippet

## Setting Root Bone of Solver

For a solver to work, it requires a Root Bone to start the solve. This will require the solver index to set the root bone. You can set the Root Bone for the solver using the following scripts:

```
1
2 # Set and get the Root Bone of the first solver.
3
4 ikr_controller.set_root_bone("pelvis", 0)
5
6 root_bone = ikr_controller.get_root_bone(0)
7
```

 Copy full snippet

## Adding/Editing/Querying IK Goals

Goals are required in order to drive an IK Solver. Goals can be added with or without a goal bone and can be renamed.

```
1
2 # Add goals to the IK Rig. You can also assign a bone during creation.
3
4 ikr_controller.add_new_goal("hand_l_goal", None)
5
6 ikr_controller.add_new_goal("TEMP_hand_r_goal", "hand_r")
7
8 # Assign a bone to an existing goal.
9
10 # You can also query for the goal in a bone and vice versa.
11
12 ikr_controller.set_goal_bone("hand_l_goal", "hand_l")
13
14 ikr_controller.get_goal_name_for_bone("hand_l")
15
16 ikr_controller.get_bone_for_goal("hand_l_goal")
17
18 # Rename goals.
19
20 ikr_controller.rename_goal("TEMP_hand_r_goal", "hand_r_goal")
```

```
21
22 # Remove a goal.
23
24 ikr_controller.remove_goal("hand_r_goal")
25
```

 Copy full snippet

You can also query for all the goals in the IK Rig.

```
1
2 # Get all goals in IK Rig.
3
4 ikr_controller.get_all_goals()
5
```

 Copy full snippet

For editing goal properties, you can get the goal object and edit any of the editor properties.

```
1
2 # Get goal object, and set the position alpha.
3
4 goal = ikr_controller.get_goal("hand_r_goal")
5
6 goal.set_editor_property("position_alpha", 0.5)
7
8 new_alpha = goal.get_editor_property("position_alpha")
9
```

 Copy full snippet

## Connecting Goals to a Solver

Now that goals have been added, you will need to connect them to a solver. You can check if a goal is connected to a solver before connecting it.


```
1
2 # Check if the goal is connected to any solver or a specific solver.
```



```

3
4 ikr_controller.is_goal_connected_to_any_solver("hand_r_goal")
5
6 ikr_controller.is_goal_connected_to_solver("hand_r_goal", 0)
7
8 # Connect or disconnect goals from a solver.
9
10 ikr_controller.connect_goal_to_solver("hand_r_goal", 0)
11
12 ikr_controller.disconnect_goal_from_solver("hand_r_goal", 0)
13

```

 Copy full snippet

## Adding/Editing/Querying Goal and Bone Settings

Once you have connected the goal to a solver, you can access the goal settings to that specific goal. Each connection between a goal and a solver will have its own specific settings.

In the following example, the `hand_r_goal` is connected to a FBIK solver and a Body Mover solver, thus it would have two different goal settings that are each associated with the corresponding solver.

```

1
2 # Get the specific settings for the goal by inputting the goal name and
  solver.
3
4 hand_r_goal_settings =
  ikr_controller.get_goal_settings_for_solver("hand_r_goal", 0)
5
6 # Edit the specific effector setting's properties.
7
8 hand_r_goal_settings.pull_chain_alpha = 0
9
10 hand_r_goal_settings.strength_alpha = 1
11
12

```

 Copy full snippet

Solvers can optionally support per-bone settings, however not all solvers will require this. Of the solvers that IK Rig ships with, only the FBlk solver has per-bone settings. The example setup for this is similar to what was previously covered with goal settings.

```
1
2 # Add bone settings to the arms.
3
4 ikr_controller.add_bone_setting("lowerarm_l", 0)
5
6 ikr_controller.add_bone_setting("lowerarm_r", 0)
7
8 ikr_controller.add_bone_setting("clavicle_l", 0)
9
10 ikr_controller.add_bone_setting("clavicle_r", 0)
11
12 # Get bone settings.
13
14 left_lowerarm_setting = ikr_controller.get_bone_settings("lowerarm_l", 0)
15
16 right_lowerarm_setting = ikr_controller.get_bone_settings("lowerarm_r", 0)
17
18 left_clav_setting = ikr_controller.get_bone_settings("clavicle_l", 0)
19
20 right_clav_setting = ikr_controller.get_bone_settings("clavicle_r", 0)
21
22 # Set the preferred angles and rotation stiffness.
23
24 left_lowerarm_setting.use_preferred_angles = True
25
26 left_lowerarm_setting.preferred_angles = unreal.Vector(0,0,90)
27
28 right_lowerarm_setting.use_preferred_angles = True
29
30 right_lowerarm_setting.preferred_angles = unreal.Vector(0,0,90)
31
32 left_clav_setting.rotation_stiffness = 1
33
34 right_clav_setting.rotation_stiffness = 1
35
```

# Checking for compatibility with other Skeletal Meshes

IK Rigs are skeleton agnostic, which means that you can share IK Rigs across similar skeleton hierarchies. With Python, you can check the mesh compatibility of an IK Rig by using a simple command.

```
1
2 # Check if this IK Rig is compatible with a defined Skeletal Mesh.
3
4 compatible_skel_mesh = unreal.load_asset(name =
    '/Game/Characters/Mannequins/Meshes/SKM_Quinn_Simple')
5
6 print(ikr_controller.is_skeletal_mesh_compatible(compatible_skel_mesh))
7
```

 Copy full snippet

# Adding, Editing, and Querying Retarget Roots and Chains

For setting up an IK Rig for a Retarget, it needs to have a Retarget Root and Retarget Chains. These can be authored using the following example Python scripts.

```
1
2 # Set or Get the Retarget Root.
3
4 ikr_controller.set_retargget_root("pelvis")
5
6 retarget_root = ikr_controller.get_retargget_root()
7
```

 Copy full snippet

For retarget chains, you can author them without inputting a start bone, end bone, or a goal. These can be added to the chain at a later time if necessary.

```
1
2 # Add a Retarget Chain with a start bone, end bone, and goal.
3
4 ikr_controller.add_retargect_chain("LeftArm", "upperarm_l", "hand_l",
  "hand_l_goal")
5
6 # Add a Retarget Chain
7
8 ikr_controller.add_retargect_chain("TEMP_RightArm", "", "", "")
9
10 # Rename a Retarget Chain.
11
12 ikr_controller.rename_retargect_chain("TEMP_RightArm", "RightArm")
13
14 # Set the Retarget Chain's start bone, end bone, and goal.
15
16 ikr_controller.set_retargect_chain_start_bone("RightArm", "upperarm_r")
17
18 ikr_controller.set_retargect_chain_end_bone("RightArm", "hand_r")
19
20 ikr_controller.set_retargect_chain_goal("RightArm", "hand_r_goal")
21
22 # Get the Retarget Chain's start bone, end bone, and goal.
23
24 ikr_controller.get_retargect_chain_start_bone("RightArm")
25
26 ikr_controller.get_retargect_chain_end_bone("RightArm")
27
28 ikr_controller.get_retargect_chain_goal("RightArm")
29
30 # Remove a Retarget Chain.
31
32 ikr_controller.remove_retargect_chain("RightArm")
33
```

 Copy full snippet

You can query for all the Retarget Chains in the IK Rig.

```
1
2 # Get all Retarget Chains in IK Rig.
```

```
3  
4 all_retargget_chains = ikr_controller.get_retargget_chains()  
5
```

 Copy full snippet