Using Python with the IK Retargeter

Using Python scripts to create and edit IK Retargeter assets to retarget character animations.

When creating and using IK Retargeter assets to retarget animations for characters and objects using IK Rigs in **Unreal Engine**, you can use custom Python scripting to control and automate IK Retargeter asset workflows.

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

Overview

An IK Retargeter asset consists of multiple parts:

- A **Source** and **Target IK Rig** to define the retarget relationship.
- A **Source** and **Target Skeletal Mesh** override from the Source and Target IK Rig.
- **Retarget Chain** mappings between the source and target, defining the relationship between each chain.
- A list of **Retarget Poses** for either the Source or Target, defining the starting pose of which the retarget will solve.

- A list of properties for the **Global**, **Retarget Root**, and each **Retarget Chain**.
- A preview scene used exclusively for an in editor preview.

The IK Retargeter data model using the following terminology:

- **Source** and **Target IK Rig**: The IK Rigs that will be used to map the Retarget Chains together in order to transfer the motion from source to target.
- **Source** and **Target Mesh**: The Skeletal Mesh in which you want to use to define the Source and Target proportions for a retarget and to access the source animations to preview.
- **Source** and **Target Retarget Pose**: The Pose in which you want either the Source or Target to use to transfer the motion. Ideally, the target should match to the pose that the source is in.
- **Global Settings**: Macro level settings that affect the entire retarget.
- **Root Settings**: Settings that affect the Retarget Root.
- **Chain Settings**: Settings that affect the specified Retarget Chain.

Accessing the IK Retargeter

The first step when scripting against any IK Retargeter asset is to gain access to the main object you'll interact with, the `IKRetargeter`. There are several ways you can gain access to the object depending on your project.

Loading an existing RTG asset

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

```

import unreal

# Ensure the file path is correct for the location of your asset in your project.

rtg = unreal.load_asset(name = '/Game/Characters/Mannequins/Rigs/RTG_Mannequin', outer = None)
```

```
```

Copy full snippet

Create a new RTG asset

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

```

# Get asset tools.

asset_tools = unreal.AssetToolsHelpers.get_asset_tools()

# Create an IK Retargeter asset in the location defined by the file path. For example, `.../Game/RTG_Mannequin`.

rtg = asset_tools.create_asset(asset_name='RTG_Mannequin', package_path='/Game/', asset_class=unreal.IKRetargeter, factory=unreal.IKRetargetFactory())

```

Copy full snippet

Editing an IK Retargeter

Preparing for an Edit by Accessing the Controller

The controller is the central object you can use to make any change to the IK Retargeter. 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 Retargeter assets:

```

# Get the IK Retargeter controller.

rtg_controller = unreal.IKRetargeterController.get_controller(rtg)

```

Copy full snippet

Assigning the Source and Target IK Rigs and Meshes to a new asset

To run a retarget, you must provide a source and a target IK Rig. In this case, we are going to create an IK Retargeter asset to retarget animations from the Manny Character to the Quinn Character, which means we can assign the `IK_Mannequin` as both the source and target.

> You can access the Manny and Quinn Skeletal Mesh characters, as well as the `IK_Mannequin` IK Rig asset, using the Third Person Template project.

```

# Load the Source and Target IK Rigs.

source_ik_rig = unreal.load_asset(name = '/Game/Characters/Mannequins/Rigs/IK_Mannequin', outer = None)

target_ik_rig = unreal.load_asset(name = '/Game/Characters/Mannequins/Rigs/IK_Mannequin', outer = None)

# Assign the Source and Target IK Rigs.

rtg_controller.set_ik_rig(unreal.RetargetSourceOrTarget.SOURCE, source_ik_rig)

rtg_controller.set_ik_rig(unreal.RetargetSourceOrTarget.TARGET, target_ik_rig)

```

Copy full snippet

We can also get the source or target IK Rig, this is useful to query for retarget chains.

```

# Get the Source and Target IK Rig assigned to the IK Retargeter asset.
```

```python

rtg_controller.get_ik_rig(unreal.RetargetSourceOrTarget.SOURCE)

rtg_controller.get_ik_rig(unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

Not only can we set the source/target IK Rigs, but we can provide overrides to the source/target preview mesh. This will override the default preview meshes provided by the IK Rigs.

```python

# Load the Skeletal Mesh.

source_skel_mesh = unreal.load_asset(name = '/Game/Characters/Mannequins/Meshes/SKM_Manny_Simple.SKM_Manny_Simple')

target_skel_mesh = unreal.load_asset(name = '/Game/Characters/Mannequins/Meshes/SKM_Quinn_Simple.SKM_Quinn_Simple')

# Assign the Source and Target Skeletal Meshes.

rtg_controller.set_preview_mesh(unreal.RetargetSourceOrTarget.SOURCE, source_skel_mesh)

rtg_controller.set_preview_mesh(unreal.RetargetSourceOrTarget.TARGET, target_skel_mesh)

# Get the Source and Target Skeletal Meshes assigned to the IK Retargeter asset.

rtg_controller.get_preview_mesh(unreal.RetargetSourceOrTarget.SOURCE)

rtg_controller.get_preview_mesh(unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

Editing and Querying Retarget Chain Mappings

Similar to the Editor, you can do an auto map for the retarget chains using Python. You can also access automap options, such as a **fuzzy string match**, an **exact string match**, or **clear maps** in total.

```

# Map chains using a fuzzy string match, which will force a remap.

rtg_controller.auto_map_chains(unreal.AutoMapChainType.FUZZY, True)

# Map chains using an exact string match, which will force a remap.

rtg_controller.auto_map_chains(unreal.AutoMapChainType.EXACT, True)

# Clear all mappings.

rtg_controller.auto_map_chains(unreal.AutoMapChainType.CLEAR, True)

```

Copy full snippet

If you don't want to do an automap, you can get or set the mapping of each target chain.

```

# Get the Source chain that is mapped to a given Target chain.

source_mapped_chain_name = rtg_controller.get_source_chain("Spine")

# Set the Source chain that the Target is mapped to, formatted in the Python script as ("Source", "Target").

rtg_controller.set_source_chain("Spine", "Spine")

```

Copy full snippet

Adding, Editing, and Querying Retarget Poses

Retarget Poses are used to determine the base pose that the source or target should retarget from. This is useful if your target character is in a different pose than your source or if the source needs to face in the proper front axis.

You can query the retarget poses for the source or target by using this command:

```python

# Get all retarget poses for the Target.

all_target_poses = rtg_controller.get_retarget_poses(unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

You can create, duplicate, rename, or remove a retarget pose, as long as you provide the retarget pose's name and specify if it is the source or target.

```python

# Create a new retarget pose.

rtg_controller.create_retarget_pose("my_new_pose", unreal.RetargetSourceOrTarget.TARGET)

# Duplicate the new pose.

rtg_controller.duplicate_retarget_pose("my_new_pose", "duplicated_pose", unreal.RetargetSourceOrTarget.TARGET)

# Rename the duplicate pose.

rtg_controller.rename_retarget_pose("duplicated_pose", "renamed_pose", unreal.RetargetSourceOrTarget.TARGET)

# Remove the duplicate pose.

rtg_controller.remove_retarget_pose("renamed_pose", unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

You can then set the current retarget pose to our newly created pose.

```
rtg_controller.set_current_retarget_pose("my_new_pose",
unreal.RetargetSourceOrTarget.TARGET)

print(rtg_controller.get_current_retarget_pose_name(unreal.RetargetSourceOrTar
```

Copy full snippet

Now that you have a new retarget pose, you can use python scripts to edit the pose. The retarget root is the only bone that can have translation and rotation offset delta values in the retarget pose. For every other bone, the retarget pose will only store the relative rotation offset delta values.

For example, your Target's retarget root needs to be raised and flipped to be in the right direction. The following python script raises it 15 units up and rotate it 90 degrees:

```
# Create a vector for the translation.

translation_offset = unreal.Vector()

translation_offset.z = 15

# Set the retarget root's translation offset.

rtg_controller.set_root_offset_in_retarget_pose(translation_offset,
unreal.RetargetSourceOrTarget.TARGET)

print(rtg_controller.get_root_offset_in_retarget_pose(unreal.RetargetSourceOr

# Create a rotator for the rotation.

rotation_offset = unreal.Rotator()

```

```
rotation_offset.roll = 90

# Set the retarget root's rotation offset, which works for any bone, and requ
bone name.

rtg_controller.set_rotation_offset_for_retarget_pose_bone("pelvis",
rotation_offset.quaternion(), unreal.RetargetSourceOrTarget.TARGET)

print(rtg_controller.get_rotation_offset_for_retarget_pose_bone("pelvis",
unreal.RetargetSourceOrTarget.TARGET))

```

Copy full snippet

Finally, you can reset a list of bones back to their original transform.

```

rtg_controller.reset_retarget_pose("my_new_pose", ["pelvis"],
unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

Auto Align Retarget Poses

```

# Auto align all bones and apply to the target retarget pose.
rtg_controller.auto_align_all_bones(unreal.RetargetSourceOrTarget.TARGET)


# Define a list of bones, align them to the source and apply to the target
retarget pose.
bones = ["spine_01", "spine_02", "spine_03"]
rtg_controller.auto_align_bones(bones,
unreal.RetargetAutoAlignMethod.CHAIN_TO_CHAIN,
unreal.RetargetSourceOrTarget.TARGET)

```

Copy full snippet

Editing and Querying Global Settings

In the Retargeter, Global Settings is where you can edit the stride adjustment properties or the different phases of a retarget, such as the Retarget Root, FK, and IK. You can query and edit these phases using the following python scripts:

```
# Get global settings, change IK, and set global settings.

global_settings = rtg_controller.get_global_settings()

global_settings.set_editor_property("enable_ik", False)

rtg_controller.set_global_settings(global_settings)
```

Copy full snippet

Editing and Querying Root Settings

In the Retargeter, Root Settings is where you would edit any properties pertaining to the Retarget Root. This would allow translation or rotation offsets to the hips, scaling the vertical or horizontal motion, or how the IK goals are affected in the vertical or horizontal planes. You can query and edit these proiperties using the following python scripts:

```
# Get root settings, raise hips by 10 units, set root settings.

root_settings = rtg_controller.get_root_settings()

root_settings.translation_offset = unreal.Vector(0,0,10)

rtg_controller.set_root_settings(root_settings)
```

Copy full snippet

Editing and Querying Chain Settings

In the Retargeter, Chain Settings is where you would edit any properties pertaining to each of the Retarget Chains. For FK chains, this would be where you change the rotation retarget mode, or For IK chains, this would be where you edit if you want the IK goal to pin to the source location, add an offset to the IK, or vertically scale the IK. You can query and edit these properties using the following python scripts:

```python

# Get the left leg chain specifically.

left_leg_chain_settings = rtg_controller.get_retarget_chain_settings("LeftLeg")

# Alternatively, you can get a chain by querying from all chains and using a one line for loop conditional.

chains = rtg_controller.get_all_chain_settings()

left_leg_chain_settings = next((chain.settings for chain in chains if chain.source_chain == "LeftLeg"), None)

# Edit the FK settings.

left_leg_chain_settings.fk.rotation_mode = unreal.RetargetRotationMode.INTERPOLATED

left_leg_chain_settings.fk.pole_vector_matching = 1

# Edit the IK settings.

left_leg_chain_settings.ik.blend_to_source = 1

left_leg_chain_settings.ik.static_offset = unreal.Vector(0,5,0)

# Edit the Speed Plant settings.
```

```

left_leg_chain_settings.speed_planting.enable_speed_planting = True

left_leg_chain_settings.speed_planting.speed_curve_name =
"ball_l_translation_speed_XYZ"

# Set the retarget chain settings.

rtg_controller.set_retarget_chain_settings("LeftLeg",
left_leg_chain_settings)

```

Copy full snippet

Post Retarget phases

```

# Load RTG_Mannequin asset

rtg = unreal.load_asset(name =
'/Game/Characters/Mannequins/Rigs/RTG_Mannequin')

rtg_controller = unreal.IKRetargeterController.get_controller(rtg)

# Get number of operations

num_ops = rtg_controller.get_num_retarget_ops()

# Get first op object

first_op = rtg_controller.get_retarget_op_at_index(0)

# Get first op index

first_op_index = rtg_controller.get_index_of_retarget_op(first_op)

# See if op is enabled

is_op_enabled = rtg_controller.get_retarget_op_enabled(first_op_index)

```

```
# Set op to be disabled

rtg_controller.set_retarget_op_enabled(first_op_index, False)

# Check result

is_op_enabled = rtg_controller.get_retarget_op_enabled(first_op_index)

rtg_controller.set_retarget_op_enabled(first_op_index, True)

# Move op in stack

rtg_controller.move_retarget_op_in_stack(first_op_index, 1)

first_op_index = rtg_controller.get_index_of_retarget_op(first_op)

# Add Curve remap op

added_curve_op_index = rtg_controller.add_retarget_op(unreal.CurveRemapOp)

added_curve_op =
rtg_controller.get_retarget_op_at_index(added_curve_op_index)

added_curve_op.set_editor_property("copy_all_source_curves", True)

curve_pair_element = unreal.CurveRemapPair()

curve_pair_element.set_editor_property("source_curve", "source")

curve_pair_element.set_editor_property("target_curve", "target")

added_curve_op.set_editor_property("curves_to_remap", [curve_pair_element])

# Remove op

rtg_controller.remove_retarget_op(added_curve_op_index)

# Add Pin Bone op

added_pin_bone_op_index = rtg_controller.add_retarget_op(unreal.PinBoneOp)

```

```
added_pin_bone_op =
rtg_controller.get_retarget_op_at_index(added_pin_bone_op_index)

translate_vector = unreal.Vector()

translate_vector.x = 120

translate_vector.y = 120

translate_vector.z = 120

offset_transform = unreal.Transform()

offset_transform.translation = translate_vector

bone_pair_element = unreal.PinBoneData()

bone_pair_element.set_editor_property("bone_to_pin", "ik_foot_root")

bone_pair_element.set_editor_property("bone_to_pin_to", "root")

added_pin_bone_op.set_editor_property("pin_type",
unreal.PinBoneType.ROTATE_ONLY)

added_pin_bone_op.set_editor_property("pin_to",
unreal.RetargetSourceOrTarget.SOURCE)

added_pin_bone_op.set_editor_property("maintain_offset", False)

added_pin_bone_op.set_editor_property("local_offset", offset_transform)

added_pin_bone_op.set_editor_property("global_offset", offset_transform)

added_pin_bone_op.set_editor_property("bones_to_pin", [bone_pair_element])

# Remove op

rtg_controller.remove_retarget_op(added_pin_bone_op_index)

# Add Root Motion generator op - auto fills names of bones

```

```
added_rm_gen_op_index =
rtg_controller.add_retarget_op(unreal.RootMotionGeneratorOp)

added_rm_gen_op =
rtg_controller.get_retarget_op_at_index(added_rm_gen_op_index)

translate_vector = unreal.Vector()

translate_vector.x = 120

translate_vector.y = 120

translate_vector.z = 120

offset_transform = unreal.Transform()

offset_transform.translation = translate_vector

added_rm_gen_op.set_editor_property("global_offset", offset_transform)

added_rm_gen_op.set_editor_property("maintain_offset_from_pelvis", False)

added_rm_gen_op.set_editor_property("propagate_to_non_retargeted_children",
True)

added_rm_gen_op.set_editor_property("root_height_source",
unreal.RootMotionHeightSource.SNAP_TO_GROUND)

added_rm_gen_op.set_editor_property("root_motion_source",
unreal.RootMotionSource.GENERATE_FROM_TARGET_PELVIS)

added_rm_gen_op.set_editor_property("rotate_with_pelvis", False)

# Remove op
rtg_controller.remove_retarget_op(added_rm_gen_op_index)

```

Copy full snippet

Batch Retarget Animations with Duplicate and Retarget

For batch processing your retargeted animations, you can run the same command as duplicate and retarget in Python.

```

# Use the asset subsystem to get asset data.

asset_subsystem = unreal.get_editor_subsystem(unreal.EditorAssetSubsystem)

# Get a list of asset data, which you can use for Anim Sequences, Anim Bluepr
Assets, and more.

assets_to_retarget = [

asset_subsystem.find_asset_data("/Game/Characters/Mannequins/Animations/Manny

asset_subsystem.find_asset_data("/Game/Characters/Mannequins/Animations/Manny

retarget_asset = unreal.load_asset(name = '/Game/Characters/Mannequins/Rigs/R

source_mesh = None # will use mesh from source ik rig

target_mesh = None # will use mesh from target ik rig

batch_op = unreal.IKRetargetBatchOperation.duplicate_and_retarget(

assets_to_retarget,

source_mesh,

target_mesh,

retarget_asset,

search = "",

replace = "",

prefix = "",
suffix = "",

remap_referenced_assets = True)
```

Copy full snippet